

Nos. 2015-1171, -1195, -1994

**UNITED STATES COURT OF APPEALS
FOR THE FEDERAL CIRCUIT**

APPLE INC., a California corporation,

Plaintiff-Cross-Appellant,

v.

SAMSUNG ELECTRONICS CO., LTD., a Korean corporation, SAMSUNG ELECTRONICS
AMERICA, INC., a New York corporation, AND SAMSUNG TELECOMMUNICATIONS
AMERICA, LLC, a Delaware limited liability company,

Defendants-Appellants.

Appeals from the United States District Court for the Northern District of
California in Case No. 12-cv-00630, Judge Lucy H. Koh.

**PLAINTIFF-CROSS APPELLANT APPLE INC.'S COMBINED PETITION
FOR PANEL REHEARING AND REHEARING EN BANC**

MARK D. SELWYN
WILMER CUTLER PICKERING
HALE AND DORR LLP
950 Page Mill Road
Palo Alto, CA 94304
(650) 858-6000

RACHEL KREVANS
ERIK J. OLSON
MORRISON & FOERSTER LLP
425 Market Street
San Francisco, CA 94105
(415) 268-7000

March 28, 2016

WILLIAM F. LEE
RICHARD W. O'NEILL
MARK C. FLEMING
LAUREN B. FLETCHER
WILMER CUTLER PICKERING
HALE AND DORR LLP
60 State Street
Boston, MA 02109
(617) 526-6000

*Counsel for Plaintiff/Cross-
Appellant Apple Inc.*

CERTIFICATE OF INTEREST

Counsel for Plaintiff/Cross-Appellant Apple Inc. certifies the following:

1. The full name of every party or amicus represented by us is:

Apple Inc.

2. The names of the real party in interest represented by us is:

Not applicable.

3. All parent corporations and any publicly held companies that own 10 percent or more of the stock of the party or amicus curiae represented by me are:

None.

4. The names of all law firms and the partners or associates that appeared for the party or amicus now represented by me in the trial court or agency or are expected to appear in this court are:

WILMER CUTLER PICKERING HALE AND DORR LLP: Thomas E. Anderson, Rebecca Bact, Heath A. Brooks, Dana O. Burwell, Anna Bonny Chauvet, Andrew J. Danford, Ronald R. Demsher (former), Mark C. Fleming, Lauren B. Fletcher, Adele R. Frankel, Sarah R. Frazier, Liv L. Herriot, Peter J. Kolovos, Gregory H. Lantier, Anna Tin-Yee Lee (former), William F. Lee, Andrew L. Liao, Cosmin Maier, Joseph J. Mueller, Richard W. O'Neill, John P. Pettit, Kevin S. Prussia, James L. Quarles, III, Katie M. Saxton, Mark D. Selwyn, Peter J. Shen (former), Michael J. Silhasek, Victor F. Souto, Thomas Sprankling, Timothy D. Syrett, Nina S. Tallon, Timothy A. Tatarka, Olga L. Tobin, S. Calvin Walden, David C. Yang, Kathryn D. Zalewski

GIBSON, DUNN & CRUTCHER LLP: Jordan H. Bekier, Brian M. Buroker, Frederick S. Chung, Emily L. Fedman (former), Megan K. Fluckiger (former), Joshua R. Furman (former), Holly A. Jones, Stephanie J. Kim, Steven S. Kim (former), Josh A. Krevitt, Jeffrey G. Lau, Jason C. Lo, Quincy Lu, H. Mark Lyon, Shannon E. Mader, Timothy W. Malone (former), Casey J. McCracken, Azar Mouzari, Mark Nolan Reiter, Jennifer J. Rho, Sarah J. Sladic (former), Rodney J. Stone, Daniel J. Thomasch, Paul

E. Torchia, Michael A. Valek, Robert Vincent, Samuel K. Whitt (former),
Minae Yu

MORRISON & FOERSTER LLP: James P. Bennett, Ruth N. Borenstein,
Brittany N. DePuy (former), Efrain Staino Flores, Richard S.J. Hung,
Michael A. Jacobs, Esther Kim, Matthew I. Kreeger, Rachel Krevans,
Kenneth A. Kuwayti, Scott F. Llewellyn, Jack W. Londen, Harold J.
McElhinny, Erik J. Olson, Mary Prendergast, Christopher L. Robinson,
Nathaniel B. Sabri, Nicole M. Smith, Christopher J. Wiener

COOLEY, GODWARD, KRONISH LLP: Benjamin George Damstedt

SKIERMONT PUCKETT LLP: Sarah Elizabeth Spires

Dated: March 28, 2016

/s/ William F. Lee

WILLIAM F. LEE

WILMER CUTLER PICKERING

HALE AND DORR LLP

60 State Street

Boston, MA 02109

(617) 526-6000

TABLE OF CONTENTS

	Page
CERTIFICATE OF INTEREST	i
TABLE OF AUTHORITIES	iv
STATEMENT OF COUNSEL UNDER FEDERAL CIRCUIT RULE 35(B)(2)	1
INTRODUCTION	1
I. THE COURT SHOULD GRANT PANEL REHEARING OR EN BANC REVIEW WITH RESPECT TO APPLE’S ’647 PATENT.....	3
A. Apple Presented Substantial Evidence Of Infringement.....	3
B. The Panel Improperly Relied On Materials Outside The Record When Reviewing The Jury’s Factual Findings.	5
C. The Panel’s Ruling Conflicts With The “Substantial Evidence” Standard And The Seventh Amendment.	7
II. THE COURT SHOULD GRANT PANEL REHEARING OR EN BANC REVIEW WITH RESPECT TO APPLE’S ’721 AND ’172 PATENTS.	11
A. Apple’s ’721 Patent	11
B. Apple’s ’172 Patent	13
C. The Panel’s Obviousness Analysis Was Contrary To Supreme Court And Federal Circuit Precedent.....	13
III. THE COURT SHOULD GRANT PANEL REHEARING WITH RESPECT TO SAMSUNG’S ’449 PATENT.	15
ADDENDUM	
CERTIFICATE OF SERVICE	
CERTIFICATE OF COMPLIANCE	

TABLE OF AUTHORITIES

CASES

	Page(s)
<i>Amadeo v. Zant</i> , 486 U.S. 214 (1988).....	1, 11
<i>Apple Inc. v. Motorola, Inc.</i> , 757 F.3d 1286 (Fed. Cir. 2014)	3, 4, 6
<i>Atlantic Thermoplastics Co. v. Faytex Corp.</i> , 5 F.3d 1477 (Fed. Cir. 1993)	1, 11
<i>Broadcom Corp. v. Qualcomm Inc.</i> , 543 F.3d 683 (Fed. Cir. 2008)	1, 7-8
<i>Group One Ltd. v. Hallmark Cards, Inc.</i> , 407 F.3d 1297 (Fed. Cir. 2005)	11
<i>KSR International Co. v. Teleflex Inc.</i> , 550 U.S. 398 (2007).....	1, 14, 15
<i>Lobatz v. U.S. West Cellular of California, Inc.</i> , 222 F.3d 1142 (9th Cir. 2000)	11
<i>Markman v. Westview Instruments, Inc.</i> , 517 U.S. 370 (1996).....	1, 11
<i>Pozen Inc v. Par Pharmaceutical, Inc.</i> , 696 F.3d 1151 (Fed. Cir. 2012)	14
<i>Reeves v. Sanderson Plumbing Products, Inc.</i> , 530 U.S. 133 (2000).....	1, 7
<i>Theme Promotions, Inc. v. News America Marketing FSI</i> , 546 F.3d 991 (9th Cir. 2008)	8
<i>Versata Software, Inc. v. SAP America, Inc.</i> , 717 F.3d 1255 (Fed. Cir. 2013)	1, 7

STATUTES AND RULES

35 U.S.C. § 282	14
Fed. R. Evid. 201(e)	11

OTHER AUTHORITIES

Dobbins, Jeffrey C., <i>New Evidence on Appeal</i> , 96 Minn. L. Rev. 2016 (2012)	11
--	----

STATEMENT OF COUNSEL UNDER FEDERAL CIRCUIT RULE 35(B)(2)

Based on my professional judgment, I believe the panel decision is contrary to the following decisions of the Supreme Court and precedents of this Court: *KSR International Co. v. Teleflex Inc.*, 550 U.S. 398 (2007); *Reeves v. Sanderson Plumbing Products, Inc.*, 530 U.S. 133 (2000); *Markman v. Westview Instruments, Inc.*, 517 U.S. 370 (1996); *Amadeo v. Zant*, 486 U.S. 214 (1988); *Versata Software, Inc. v. SAP America, Inc.*, 717 F.3d 1255 (Fed. Cir. 2013); *Broadcom Corp. v. Qualcomm Inc.*, 543 F.3d 683 (Fed. Cir. 2008); and *Atlantic Thermoplastics Co. v. Faytex Corp.*, 5 F.3d 1477 (Fed. Cir. 1993).

Dated: March 28, 2016

/s/ William F. Lee

WILLIAM F. LEE

INTRODUCTION

After a thirteen-day trial and three-and-a-half days of deliberation, a jury found that Samsung's smartphones infringed three Apple patents, confirmed the validity of Apple's asserted patents, and awarded damages of nearly \$120 million. The issues on appeal were primarily factual and subject to deferential substantial evidence review. Yet, in an unprecedented decision, the panel reversed nearly every aspect of the verdict that favored Apple. The panel reached that result by deciding the case on a different record than the jury had before it, and by shifting Samsung's burden to prove invalidity to require instead that Apple prove validity.

With respect to Apple’s U.S. Patent No. 5,946,647, the panel reversed the jury’s infringement verdict (which accounted for over \$98 million in damages) by relying on dictionary definitions, an encyclopedia, a textbook, and an unrelated patent—none of which was of record and that the panel appears to have located only through independent research. Equally troubling, the panel selectively relied on portions of these non-record materials, while omitting other statements in the same materials that *support* the jury’s infringement verdict. Such appellate fact finding is contrary to the “substantial evidence” standard and violates Apple’s Seventh Amendment right to have a jury decide the factual issue of infringement.

The panel also reversed the jury’s verdict of no invalidity for Apple’s U.S. Patent Nos. 8,046,721 and 8,074,172. For each, the panel found a “strong” case of obviousness because the prior art (supposedly) disclosed all claimed elements. But the panel impermissibly excused Samsung from its duty to show that a person of ordinary skill in the art would have had a reason to combine the asserted references—even though the evidence permitted the jury to find, as a factual matter, that a skilled artisan would *not* have made the proposed combinations. The panel further erred by placing the burden on Apple to disprove obviousness, and by dismissing Apple’s evidence of objective indicia under an unduly rigid standard.

Finally, the panel took a different, but equally erroneous, approach to Samsung’s U.S. Patent No. 6,226,449—an approach that cannot be reconciled with

the panel’s reversal on Apple’s patents. Specifically, the panel misapprehended or overlooked that there was no evidence that Apple’s products contain the claimed “recording circuit [that] records ... image signals with classification data”—and the panel identified none. Panel rehearing is appropriate to correct this error.

I. THE COURT SHOULD GRANT PANEL REHEARING OR EN BANC REVIEW WITH RESPECT TO APPLE’S ’647 PATENT.

A. Apple Presented Substantial Evidence Of Infringement.

Computer data often contains different types of “structures” (e.g., phone numbers, email addresses) that users may want to use to perform specific functions. Manually identifying and using that information can be “tedious” and “disruptive,” because it requires locating the information and copying or entering it into another program. A594. The invention of Apple’s ’647 patent automatically “detects” structures in text, generates “links” to specific actions that can be performed for the detected structure (e.g., dial a phone number, compose an email), and provides a pop-up menu for the user to select a linked action. A597; *see* Op. 4.

Asserted claim 9 requires an “analyzer server” that performs the claimed “detecting” and “linking” functions. A597; *see* Op. 6-7. Neither party sought construction of “analyzer server” during *Markman*. On the last scheduled day of trial testimony, however, this Court issued a decision in *Apple Inc. v. Motorola, Inc.*, 757 F.3d 1286 (Fed. Cir. 2014), which construed “analyzer server” in the ’647 patent to mean “a server routine *separate from* a client that receives data having

structures from the client.” *Id.* at 1304.¹ Following the *Motorola* decision, the district court extended trial by a day and allowed the parties to recall their experts to provide testimony applying this Court’s claim construction. The district court also instructed the jury to apply the *Motorola* construction. A13027; A13172.

Apple’s expert Dr. Todd Mowry testified that two applications on Samsung’s smartphones—“Browser” and “Messenger”—send data to an “analyzer server” that detects structures and links them to specified actions. A13030-40; A10853-64. Based upon his review of Samsung’s source code (A13044-45), Dr. Mowry explained that those applications are “clients” and identified code in certain shared libraries as the separate “server routines” that receive data having structures from the Browser and Messenger clients. A13030-35; A50865; A50867-68.

During his recall testimony, Dr. Mowry explained how “shared library code” in Samsung’s phones is an “analyzer server” under the *Motorola* construction. A13029-45.² He walked through the relevant source code and described how it performs the claimed “detecting” and “linking” functions. A13030-35; A50864-

¹ Emphases are added unless indicated otherwise.

² The panel suggested that Dr. Mowry’s recall testimony was inconsistent with his prior testimony. Op. 9-10. It was not. In his initial testimony, Dr. Mowry explained how the shared library code was an “analyzer server” under that term’s plain meaning. A10853-64; A12800-01. On recall, Dr. Mowry explained how the code met the *added* requirement of being “separate from” the client, as required by *Motorola*. A13035-37. It was *Samsung’s* expert who the trial court reprimanded for “improper” and “misleading” testimony seeking to portray his recall testimony as consistent with his prior opinions. Dist. Ct. ECF 1928, Trial Tr. 3055-77.

68. He further testified that Samsung's shared library code exists and runs separately from the client applications it serves. A13064 ("Shared libraries don't run as part of an application."); A13036 (shared library code is "definitely separate from the applications"); A13035-37 (shared library code is "developed independently of" the Browser and Messenger applications, stored "in a different part of the [memory] address space," and "designed to be reused across different applications"). As Dr. Mowry told the jury, different applications use the shared library code "without actually duplicating the code"; each application "goes to the code where it is and uses it there, and it does that each time that it accesses the code." A13037. Based on this evidence, the jury found that Samsung infringed.³

B. The Panel Improperly Relied On Materials Outside The Record When Reviewing The Jury's Factual Findings.

Samsung did not appeal the "analyzer server" claim construction. Nor could it have, since the district court instructed the jury to apply this Court's *Motorola* construction (A13027; A13172), to which both parties agreed. Instead, Samsung raised only a substantial evidence challenge, arguing that the trial record did not support the jury's finding that the shared library code was "separate from" the Browser and Messenger applications. Blue Br. 16-25; Yellow Br. 5-15.

³ This verdict was reached by a sophisticated jury. In fact, the foreperson was a former IBM executive responsible for "the technical direction for a significant amount of IBM's large systems software." Dist. Ct. ECF 1621, Trial Tr. 195-96.

A panel of this Court reversed, but did so only by relying on information outside the record to conclude that Dr. Mowry's testimony was "not sufficient evidence to allow a jury to conclude that the Samsung software met the 'analyzer server' limitation." Op. 10. The panel did this in two ways.

First, although Dr. Mowry applied the plain meaning of "server" (which had not been separately construed and which this Court previously noted "entails a client-server relationship," *Motorola*, 757 F.3d at 1304), the panel relied on non-record sources to make its own finding as to that word's plain meaning. Specifically, the panel ruled that "servers" must be "'standalone' programs that run separately." Op. 13; *see* Op. 10. The panel relied upon descriptions of "servers" in Encyclopedia of Computer Science 215 (4th ed. 2000); Montgomery, Object-Oriented Information Engineering 265 (1994); and U.S. Patent No. 5,546,583, issued to IBM in 1996. Op. 9-11. None of those sources was in the trial court record or raised by either party on appeal, nor were they cited in *Motorola*.

Second, the panel also looked to external sources in considering the factual question whether Samsung's phones met the "analyzer server" limitation. The panel relied upon a dictionary definition and an encyclopedia entry to conclude that "the software library program" in Samsung's phones "runs as part of the client program." Op. 8-9 (citing Dictionary of Computing 391 (4th ed. 1996) (Add. Tab 5)); *see also* Op. 8 (citing Encyclopedia of Computer Science 1620 (4th ed. 2000)

(Add. Tab 2)). Again, these sources were not in the evidentiary record or raised by either party on appeal, and were apparently the result of the panel’s own research.

From all this, the panel concluded that no reasonable jury could have found infringement of the “analyzer server” limitation because “the Samsung software library programs are not ‘standalone’ programs that run separately.” Op. 13.⁴

C. The Panel’s Ruling Conflicts With The “Substantial Evidence” Standard And The Seventh Amendment.

The jury’s finding that Samsung’s phones satisfied the “analyzer server” limitation was “a pure factual issue,” which should have been reviewed for substantial evidence. *Versata Software, Inc. v. SAP Am., Inc.*, 717 F.3d 1255, 1262 (Fed. Cir. 2013). As the Supreme Court has made clear, substantial evidence review occurs on the trial record—and only the trial record. *Reeves v. Sanderson Plumbing Prods., Inc.*, 530 U.S. 133, 150-151 (2000) (“[On] motion for judgment as a matter of law, the court should review all of the evidence *in the record* ... [and] draw all reasonable inferences in favor of the nonmoving party.”). Every circuit, including this Court, follows this rule. *E.g., Broadcom Corp. v. Qualcomm*

⁴ The panel also cited statements from Apple’s counsel regarding whether the “analyzer server” “has to be run separately from the client.” Op. 11 (quoting Oral Arg. 29:28-34). But Apple’s counsel was clear that “[t]here is no requirement that [the analyzer server] be standalone. The requirement is that it be a separate routine.” Oral Arg. 28:52-58. And counsel’s statement that the analyzer server “run[s] separately” was accompanied by an explanation that the trial record (and the jury’s verdict) showed that Samsung’s devices met that limitation. Oral Arg. 29:37-40 (“It’s a routine that runs separately and that’s what Dr. Mowry said.”); *see* A13064 (Mowry) (“Shared libraries *don’t run as part of an application.*”).

Inc., 543 F.3d 683, 696 (Fed. Cir. 2008) (“In reviewing factual issues for substantial evidence, the inquiry is whether a reasonable jury, *given the record before it viewed as a whole*, could have arrived at the conclusion it did.”); *Theme Promotions, Inc. v. News Am. Mktg. FSI*, 546 F.3d 991, 997 (9th Cir. 2008) (affirming jury verdict because “supported by substantial evidence *in the record*”).

The panel departed from that fundamental rule. The panel looked to a textbook, encyclopedia entries, and an unrelated patent to create its own plain meaning of “server” as requiring a “standalone” program. Op. 8-11. The panel used those materials to override the evidence presented to the jury—including Dr. Mowry’s testimony, based upon his years of experience as a computer scientist, that the plain meaning of “server” includes shared libraries. A13060 (“There are multiple ways to implement a client server. One way is with shared libraries.”).⁵

The panel also relied on dictionary and encyclopedia entries to inform its understanding of how the shared library code *in Samsung’s phones* works. Op. 8-9. The panel treated that extrinsic material as a substitute for record evidence, even where it was flatly contrary to the evidence before the jury. *Compare* Op. 9 (“[T]he software library program runs as part of the client program.” (citing

⁵ Even under the panel’s plain meaning of “server,” there was substantial evidence of infringement. While the panel said there was no “testimony where [Dr. Mowry] stated that the library programs run separately” (Op. 11, 13), he testified that “[s]hared libraries don’t run as part of an application.” A13064 (cited Red Br. 19).

Dictionary of Computing 391 (Add. Tab 5)), with A13064 (Mowry) (“Shared libraries don’t run as part of an application.”) (not cited by panel).

To compound the error of relying on materials the jury never saw, the panel quoted only selective portions of its new materials, while omitting surrounding text that contradicted the panel’s conclusions. For example, the panel quoted the Montgomery text for the proposition that “[a] client/server relationship assumes a ‘clean separation of functions,’” which the panel interpreted to exclude reliance on shared library code. Op. 10 n.5; see Op. 11-12. Yet the next sentence in the Montgomery text states that a “server” may “regulate access to shared resources”:

Shared resources

- A **server** can service many clients at the same time and regulate access to **shared resources**.

Montgomery, *supra*, at 265 (Add. Tab 3). The cited patent has a similar statement. U.S. Patent No. 5,546,583 (1:28-29) (“A **server** may service many clients at the same time and regulate their access to **shared resources**.”) (Add. Tab 4). That is precisely how Dr. Mowry explained “shared libraries”: “[S]hared libraries in general are designed so that multiple applications can access them and use them.” A13037. The portions of the external sources that the panel omitted are thus consistent with Dr. Mowry’s testimony, which confirmed that a “shared library” may be a “server” under that word’s plain meaning. A13060 (“There are multiple ways to implement a client server. One way is with shared libraries.”).

The panel's use of external sources to inform its view of how Samsung's phones work was equally troubling. The panel quoted the last sentence from the Dictionary of Computing's definition of "program library (software library)" to suggest that shared library code is "incorporated in a user's program." Op. 9. But the full quotation confirms that shared libraries may include programs made available for "***common use*** within some environment," including examples (*e.g.*, "compilers") that would satisfy the panel's requirement of a "standalone" program:

A collection of programs and packages that are made available for common use within some environment; individual items need not be related. A typical library might contain compilers, utility programs, packages for mathematical operations, etc. Usually it is only necessary to reference the library program to cause it to be automatically incorporated in a user's program.

Dictionary of Computing 391 (Add. Tab 5). The portion omitted by the panel was consistent with Dr. Mowry's testimony—which the jury was entitled to credit but the panel rejected (Op. 11)—that applications use Samsung's shared library code "without actually duplicating the code"; each application "goes to the code where it is and uses it there, and it does that each time that it accesses the code." A13037.

Apple had no opportunity to address the new material that the panel relied upon, let alone to have a jury weigh it. The jury was admonished that "you must decide the case solely on the evidence before you." Dist. Ct. ECF 1847 (Instruction No. 1). A court cannot overturn a verdict by considering materials the jury was forbidden from considering. Yet that is what happened here. *See*

Amadeo v. Zant, 486 U.S. 214, 228 (1988) (criticizing “impermissible appellate factfinding”); *Atlantic Thermoplastics Co. v. Faytex Corp.*, 5 F.3d 1477, 1479 (Fed. Cir. 1993) (“Fact-finding by the appellate court is simply not permitted.”).⁶

Unless corrected, the panel’s decision will undermine the jury’s role and leave verdicts susceptible to reversal based on facts the jury never saw. *See Dobbins, New Evidence on Appeal*, 96 Minn. L. Rev. 2016, 2016 (2012) (“Appellate review is limited ... to consideration of the factual record as established in the trial court. This limitation ... respects trial processes for presenting, evaluating, and admitting evidence, protects the fairness of the system with respect to the parties, and helps ensure accuracy[.]”). The panel’s ruling also undermines Apple’s Seventh Amendment right to have a jury—not an appellate court—decide the facts. *See Markman v. Westview Instruments, Inc.*, 517 U.S. 370, 376-377 (1996).

II. THE COURT SHOULD GRANT PANEL REHEARING OR EN BANC REVIEW WITH RESPECT TO APPLE’S ’721 AND ’172 PATENTS.

A. Apple’s ’721 Patent

The ’721 patent relates to Apple’s iconic “slide to unlock” feature, which prevents the inadvertent activation of a touchscreen device by requiring users to

⁶ The panel’s action cannot be justified as “judicial notice,” which the panel did not reference, because Samsung could have submitted the materials to the trial court in the first place. *Lobatz v. U.S. West Cellular of Cal., Inc.*, 222 F.3d 1142, 1148 (9th Cir. 2000); *Group One Ltd. v. Hallmark Cards, Inc.*, 407 F.3d 1297, 1306 (Fed. Cir. 2005) (when “unfairness results,” it is “improper for the district court to take judicial notice after the verdict”); *see also* Fed. R. Evid. 201(e).

slide an image across the screen to unlock the device. At trial, Samsung argued obviousness based on two references. The first (“Neonode”) describes a mobile phone, but does not disclose the claim requirement of an “unlock image” that is moved from one predefined location to another to unlock the device. Op. 17. To fill that gap, Samsung pointed to a second reference (“Plaisant”), which describes a wall-mounted device to control home appliances like air-conditioning units and heaters. A12876; A20741-43. Plaisant explains that sliders are “not preferred,” “more complex,” and “more difficult to implement” than alternative toggle controls. A20743. Apple’s expert testified that a skilled artisan would not have thought to combine Neonode with Plaisant, particularly given Plaisant’s criticisms of sliders. A12877-78. The jury agreed and found claim 8 not invalid.

The panel reversed, emphasizing that the references “disclose[] all of the claimed features.” Op. 19. Rather than requiring Samsung to show that a skilled artisan would have combined the references, the panel required Apple to show that “a skilled artisan would *not* have had the motivation to combine Neonode and Plaisant.” Op. 19. The panel also dismissed Apple’s objective indicia, including long-felt need, industry praise, copying, and commercial success. Op. 24-28. The panel adopted unduly rigid rules, dismissing Samsung’s praise for the invention because it was not public and refusing to credit evidence of Samsung’s copying because it included “a feature shown in the Plaisant prior art.” Op. 27 & n.10.

B. Apple’s ’172 Patent

The ’172 patent relates to “auto-correct” software that automatically corrects typing errors entered on a touchscreen device. The district court granted summary judgment of infringement. At trial, Samsung argued obviousness based on two references. It was undisputed that one reference (“Robinson”) fails to disclose a key claim element—“displaying and replacing an incorrectly typed word in a first area.” Op. 32. As Apple’s expert testified, the second reference (“Xrgomics”) also does not disclose that element. Instead of teaching text *correction* which replaces text that has already been typed, Xrgomics describes text *completion* where the software suggests alternatives for adding letters to complete a word as it is being typed. A12916-17 (cited Red Br. 36-37). The jury found claim 18 not invalid.

The panel reversed. With no discussion of whether Xrgomics discloses the “replacing” claim element, the panel found the combination “results in Apple’s invention.” Op. 32-33. The panel’s analysis rested on its (mistaken) belief that the references taught all claim elements without requiring Samsung to prove a reason to combine them. Op. 35. The panel also dismissed Apple’s evidence of objective indicia, including Samsung’s copying and commercial success. Op. 34-35.

C. The Panel’s Obviousness Analysis Was Contrary To Supreme Court And Federal Circuit Precedent.

In *KSR*, the Supreme Court emphasized that “a patent composed of several elements is not proved obvious merely by demonstrating that each of its elements

was, independently, known in the prior art.” 550 U.S. at 418. Rather, to demonstrate obviousness, it is necessary to show that “there was an apparent reason to combine the known elements in the fashion claimed by the patent.” *Id.*

The panel applied the very rule that *KSR* condemned—finding a “strong” obviousness case because all claim elements were (supposedly) known. Op. 35 (“[W]e find that Samsung presented a strong case of obviousness, showing that every element ... was present in the prior art.”); Op. 19, 24 (finding “strong” obviousness case because “Plaisant, when combined with Neonode, discloses all of the claimed features”). This emphasis on whether the elements were known, rather than whether Samsung proved a skilled artisan would have had reason to combine them, has sweeping consequences. “[I]nventions ... rely upon building blocks long since uncovered, and claimed discoveries almost of necessity will be combinations of what, in some sense, is already known.” *KSR*, 550 U.S. at 418-419.

Rather than require Samsung to show a reason to combine the references, the panel gave Apple the burden of proving that “a skilled artisan would *not* have had the motivation to combine” the references. Op. 19; *see* Op. 33. That reasoning is contrary to the statutory presumption of validity. *See Pozen Inc v. Par Pharm.*, 696 F.3d 1151, 1159-1160 (Fed. Cir. 2012); 35 U.S.C. § 282.

Additionally, the panel adopted an extremely rigid approach to Apple’s evidence of objective indicia—for example, by dismissing evidence of long-felt

need as just the opinion of “a single expert” (Op. 26) and by rejecting evidence of Samsung’s own praise for the invention because it was not public (Op. 27 n.10). Those rigid distinctions are not supported in the case law and are contrary to the “flexible approach” to obviousness mandated by *KSR*. 550 U.S. at 415.

III. THE COURT SHOULD GRANT PANEL REHEARING WITH RESPECT TO SAMSUNG’S ’449 PATENT.

Samsung’s ’449 patent describes a digital camera that includes hardware and software for recording, classifying, and searching both still images (photos) and moving images (videos). Op. 43. The jury found that Apple infringed claim 27 and awarded \$158,400 in damages. *Id.* The panel affirmed. Op. 43-45.

Claim 27 requires a “recording circuit” that “records each ... image signal[] with classification data.” Op. 44. The panel stated that “Samsung presented testimony that the Apple products record images with classification data.” Op. 45. But there was no such testimony or evidence, and the panel cited none. The only testimony that Samsung identified (Yellow Br. 57) did not even address whether Apple’s products satisfy the “records ... with classification data” limitation.⁷ The panel should grant rehearing and reverse the infringement judgment for this patent.

⁷ Two Samsung citations refer to the ’449 patent, not Apple’s products (A12602:9-13; A12608:11-13); one discusses how to delete a video (a “moving image”), rather than how to “record[]” the “still images” that are the subject of the “records ... with classification data” limitation (A12611:7-13); and one addresses only a separate limitation (“a list of classifications as a classification mode” by displaying albums) (A12625:9-15).

Respectfully submitted,

/s/ William F. Lee

WILLIAM F. LEE
RICHARD W. O'NEILL
MARK C. FLEMING
LAUREN B. FLETCHER
WILMER CUTLER PICKERING
HALE AND DORR LLP
60 State Street
Boston, MA 02109
(617) 526-6000

*Counsel for Plaintiff/Cross-
Appellant Apple Inc.*

MARK D. SELWYN
WILMER CUTLER PICKERING
HALE AND DORR LLP
950 Page Mill Road
Palo Alto, CA 94304
(650) 858-6000

RACHEL KREVANS
ERIK J. OLSON
MORRISON & FOERSTER LLP
425 Market Street
San Francisco, CA 94105
(415) 268-7000

March 28, 2016

ADDENDUM

TABLE OF CONTENTS

	Tab
<i>Apple Inc. v. Samsung Electronics Co.</i> , Slip Opinion, No. 15-1171 (Fed. Cir. Feb. 26, 2016)	Tab 1
<u>Encyclopedia of Computer Science</u> , pp. 215-218, 1620-1624 (4th ed. 2000)	Tab 2
Montgomery, Stephen L., <u>Object-Oriented Information Engineering: Analysis, Design, and Implementation</u> , 264-271 (1994).....	Tab 3
United States Patent No. 5,546,583	Tab 4
<u>Dictionary of Computing</u> , p. 391 (4th ed. 1996)	Tab 5

TAB 1

**United States Court of Appeals
for the Federal Circuit**

APPLE INC., A CALIFORNIA CORPORATION,
Plaintiff-Cross-Appellant

v.

**SAMSUNG ELECTRONICS CO., LTD., A KOREAN
CORPORATION, SAMSUNG ELECTRONICS
AMERICA, INC., A NEW YORK CORPORATION,
SAMSUNG TELECOMMUNICATIONS AMERICA,
LLC, A DELAWARE LIMITED LIABILITY
COMPANY,**
Defendants-Appellants

2015-1171, 2015-1195, 2015-1994

Appeals from the United States District Court for the
Northern District of California in No. 5:12-cv-00630-LHK,
Judge Lucy H. Koh.

Decided: February 26, 2016

WILLIAM F. LEE, Wilmer Cutler Pickering Hale and
Dorr LLP, Boston MA, argued for plaintiff-cross-
appellant. Also represented by DANA OLCOTT BURWELL,
ANDREW J. DANFORD, MARK CHRISTOPHER FLEMING,
LAUREN B. FLETCHER, SARAH R. FRAZIER, RICHARD WELLS

O'NEILL; MARK D. SELWYN, Palo Alto, CA; THOMAS GREGORY SPRANKLING, Washington, DC; RACHEL KREVANS, Morrison & Foerster LLP, San Francisco, CA; ERIK JEFFREY OLSON, Palo Alto, CA.

KATHLEEN M. SULLIVAN, Quinn Emanuel Urquhart & Sullivan, LLP, New York, NY, argued for defendants-appellants. Also represented by WILLIAM ADAMS, DAVID MICHAEL COOPER; BRIAN COSMO CANNON, KEVIN P.B. JOHNSON, VICTORIA FISHMAN MAROULIS, Redwood Shores, CA; JOHN B. QUINN, SCOTT L. WATSON, MICHAEL THOMAS ZELLER, Los Angeles, CA.

Before PROST, *Chief Judge*, DYK, and REYNA, *Circuit Judges*.

DYK, *Circuit Judge*.

The current appeal results from a patent infringement suit and countersuit between Apple Inc. (“Apple”) and Samsung Electronics Co., Ltd., Samsung Electronics America, Inc., and Samsung Telecommunications America, LLC (collectively, “Samsung”). Apple alleged infringement of five U.S. patents that it owns: U.S. Patent Nos. 5,946,647 (the ‘647 patent), 6,847,959 (the ‘959 patent), 7,761,414 (the ‘414 patent), 8,046,721 (the ‘721 patent), and 8,074,172 (the ‘172 patent). After a jury trial, the district court entered a judgment awarding Apple \$119,625,000 in damages and ongoing royalties¹ for infringement of the ‘647 patent, the ‘721 patent, and the ‘172 patent. The jury found that Samsung had not infringed the ‘959 patent and the ‘414 patent. The district court entered judgment accordingly.

¹ The district court determined that Apple was entitled to ongoing royalties but did not quantify the amount.

Samsung's countersuit alleged infringement of two patents that it owns: U.S. Patent Nos. 5,579,239 (the '239 patent) and 6,226,449 (the '449 patent). The jury found Apple had infringed the '449 patent and awarded \$158,400 in damages but found that Apple had not infringed the '239 patent. The district court entered judgment in accordance with the jury verdict.

Both Apple and Samsung appeal. With regard to Apple's '647 patent, we reverse the district court's denial of Samsung's motion for judgment as a matter of law (JMOL) of non-infringement and find that Apple failed to prove, as a matter of law, that the accused Samsung products use an "analyzer server" as we have previously construed that term. We also reverse the district court's denial of JMOL of invalidity of Apple's '721 and '172 patents, finding that the asserted claims of both patents would have been obvious based on the prior art. We affirm the judgment of non-infringement of Apple's '959 and '414 patents, affirm the judgment of infringement of Samsung's '449 patent, and affirm the judgment of non-infringement of Samsung's '239 patent. In light of these holdings, we need not address the other issues on this appeal. Accordingly, we affirm-in-part and reverse-in-part.

BACKGROUND

This is our third appeal in this case. In the first appeal, we reversed the district court's order granting a preliminary injunction enjoining Samsung from selling one of its smartphones in the United States based on a patent no longer at issue in this case. *Apple Inc. v. Samsung Elecs. Co.*, 695 F.3d 1370 (Fed. Cir. 2012) ("*Apple I*"). In the second appeal, we vacated a district court remedial order denying Apple's request for a permanent injunction that would have enjoined Samsung from "making, using, selling, developing, advertising, or importing into the

United States software or code capable of implementing the infringing features [of the '647, the '721, and the '172 patents] in its products.” *Apple Inc. v. Samsung Elecs. Co.*, 809 F.3d 633, 638 (Fed. Cir. 2015).² The district court decision and our reversal addressed the appropriateness of injunctive relief for assumed infringement. That decision did not address or resolve the merits of the underlying case that is now before us. In this third appeal, we confront the core infringement and invalidity issues with respect to the asserted patents.

I

Apple filed suit against Samsung on February 8, 2012, asserting infringement of eight patents, including the five that are relevant for this appeal. Samsung answered, contesting infringement and alleging invalidity of the asserted patents. In addition, Samsung countersued Apple for infringement of eight patents that it owns, including the two relevant for the current appeal. Before trial, the parties reduced the number of asserted claims, with Apple maintaining infringement as to five patents and Samsung maintaining allegations of infringement of two patents.

The five Apple patents involved at trial and on appeal cover various aspects of the operation of smartphones. The '647 patent covers software to detect “structures,” such as a phone number, in text and to turn those structures into links, thus allowing a user to “click” on the structure to take an action (such as making a phone call) rather than having to copy and paste the structure into another application. The '721 patent is directed to the

² On January 18, 2016, the district court entered the requested injunction, which was automatically stayed for 30 days.

iPhone's "slide to unlock" feature, where a user can slide a moving image across the screen of the phone with his finger to unlock the phone. The '172 patent covers "auto-correct" software on the phone that automatically corrects typing errors. The '959 patent claims "Universal Search," where a user can, from a single search term, find results both from applications on the phone and from the Internet. Lastly, Apple's '414 patent covers "Background Sync" software that synchronizes information on the phone with other devices while the user is using the phone.

As to Samsung's patents, the '449 patent covers camera systems for compressing, decompressing, and organizing digital photos and videos. The '239 patent covers systems for compressing and transmitting videos.

After a 13-day trial, the jury found all asserted claims of the Apple patents not invalid and awarded Apple \$119.6 million for infringement of the asserted claims of the '647, '721, and '172 patents.³ The jury, however, found that Samsung had not infringed Apple's '414 patent or Apple's '959 patent. Additionally, the jury found that Apple had infringed the asserted claim of the '449 patent, awarding Samsung \$158,400 in damages, but found Samsung's '239 patent not infringed. The district court entered judgment.

We have jurisdiction pursuant to 28 U.S.C. § 1295(a)(1). We review a district court's order granting or denying JMOL under the standard applied by the regional circuit. In the Ninth Circuit, the review is de novo, and the court views the evidence in the light most

³ The jury found the asserted claims of the '647 and the '721 patents infringed, and the district court had previously entered summary judgment of infringement of the asserted claim of the '172 patent.

favorable to the jury verdict. *See Amarel v. Connell*, 102 F.3d 1494, 1521 (9th Cir. 1996).

DISCUSSION

I. The Apple '647 Patent

Apple asserted infringement of claim 9 of the '647 patent. The jury found that Samsung infringed and awarded Apple \$98,690,625. The district court denied JMOL of non-infringement.

Samsung argues that the district court erred in not granting its motion for JMOL of non-infringement. The '647 patent “discloses a system for recognizing certain structures (such as a telephone number) on a touchscreen and then linking certain actions (such as calling the telephone number) to the structure. For example, a user may be able to call or save a phone number it has received via text message or email simply by touching the number on the screen of its device.” *Apple Inc. v. Motorola, Inc.*, 757 F.3d 1286, 1304 (Fed. Cir. 2014) (“*Motorola*”). Asserted claim 9 depends on claim 1. Claim 1 reads:

A computer-based system for detecting structures in data and performing actions on detected structures, comprising:

an input device for receiving data;

an output device for presenting the data;

a memory storing information including program routines including

an analyzer server for detecting structures in the data, and for linking actions to the detected structures;

a user interface enabling the selection of a detected structure and a linked action; and

an action processor for performing the selected action linked to the selected structure; and

a processing unit coupled to the input device, the output device, and the memory for controlling the execution of the program routines.

'647 patent, col. 7 ll. 9–24 (emphasis added). Claim 9 adds an additional limitation, “wherein the user interface enables selection of an action by causing the output device to display a pop-up menu of the linked actions.” *Id.* at ll. 53–55.

Samsung contends that Apple failed to produce any evidence from which a reasonable jury could conclude that Samsung’s allegedly infringing phones practiced the “analyzer server” limitation.⁴

Before trial, neither party sought construction of “analyzer server,” agreeing that it should be given its ordinary meaning. However, on the last scheduled day of trial, we issued a decision in another case construing this term in the same claim at issue here. *See Motorola*, 757 F.3d at 1304. The district court adopted our construction and allowed each party to recall its expert witnesses to

⁴ Samsung also maintains that Apple failed to provide any evidence that the accused software in the Samsung devices practiced the “linking actions to the detected structures” limitation. In light of our holding as to the “analyzer server” limitation, we need not address this issue.

address whether the allegedly infringing devices met the limitation under our new construction. The district court then allowed the case to proceed to the jury.

In the *Motorola* case, we construed “analyzer server” to mean “a server routine separate from a client that receives data having structures from the client.” *Id.* We found that the “plain meaning of ‘server,’ when viewed from the perspective of a person of ordinary skill in the art, entails a client-server relationship. Consistent with this perspective, the specification discloses an analyzer server that is separate from the application it serves.” *Id.* We rejected Apple’s proposed construction—“a *program routine(s)* that receives data, uses patterns to detect structures in the data and links actions to the detected structures”—and Apple’s arguments that “the analyzer server need not be ‘separate from a client.’” *Id.* We found that the proposed construction and argument “conflict[] with the claim language by ignoring the claim term ‘server.’” *Id.* at 1305. In other words, Apple tried to “take[] the claim text and remove[] the ‘analyzer server,’ leaving the rest basically unchanged.” *Id.* Our construction required that the “analyzer server” be a piece of software that runs separately, receives data from a client application, performs the “detecting” and “linking” steps, and then returns that data to the client application. *Id.* at 1304–05.

Here, Apple accused two applications on Samsung devices of infringing claim 9: the Browser application (the web browser) and the Messenger application (used for text messaging). For these applications, Apple asserted that pieces of software code stored in shared program libraries were the “analyzer server” that performed the “detecting” and the “linking” functions. A “program library is a collection of computer programs for a particular application.” *Software Libraries*, Encyclopedia of Computer Science 1620 (4th ed. 2000). Libraries contain collections

of programs to perform specific operations common to many different applications. *Id.* As the name implies, a client program can go to the shared program library and “borrow” (i.e., use) code from the library to perform a specific needed task rather than having to program that functionality into the client program. In other words, the software library program runs as part of the client program. *See Program library (software library)*, Dictionary of Computing 391 (4th ed. 1996) (“Usually it is only necessary to reference the library program to cause it to be automatically *incorporated in* a user’s program.”) (emphasis added). In a client-server implementation, as our previous opinion recognized, *Motorola*, 757 F.3d at 1304–05, the client sends information to a separately-running independent program which then performs a task using that information and sends information back to the client program. *See Client-Server Computing*, Encyclopedia of Computer Science 215 (4th ed. 2000).

There can be no question that before the last day of trial, Apple tried its case based on the claim construction that we rejected in *Motorola*. Apple’s expert explicitly testified that the claim language covered any “piece of software that performs these functions,” J.A. 10896, and that the claim language did not require software that could be used across different applications. In other words, Apple’s expert, prior to the last day of trial, testified that the “analyzer server” need not be a separate piece of software that runs on its own.

On the last day of trial, Apple recalled the same witness to testify that the accused devices infringed even under our new claim construction. He testified that the accused software was a separate “analyzer server” because the Samsung application (i.e., Messenger) “goes to the code where it is and uses it there, and it does that each time it accesses the code.” J.A. 13037. He also testified that these shared library programs were “defi-

nitely separate from the applications” because they were stored in a different part of memory, they received data from the Messenger and Browser applications, and they were developed independently of the Browser and Messenger applications. J.A. 13035–36.

However, this testimony is not sufficient evidence to allow a jury to conclude that the Samsung software met the “analyzer server” limitation. Our previous construction required more than just showing that accused software was stored in a different part of the memory and was developed separately. We found that the “analyzer server” limitation is a separate structural limitation and must be a “server routine,” consistent with the “plain meaning of ‘server’.” *Motorola*, 757 F.3d at 1304. That is, it must run separately from the program it serves.⁵ See

⁵ Specifically, we found that the “analyzer server” had to involve a “client-server relationship.” *Motorola*, 757 F.3d at 1304. “Client-server computing is a distributed computing model in which client applications request services from server processes.” *Client-Server Computing, Encyclopedia of Computer Science* 215 (4th ed. 2000). The “client application is a process or program that sends messages to a server Those messages request the server to perform a specific task” *Id.* “The server process or program listens for client requests that are transmitted Servers receive those requests and perform actions such as database queries and reading files.” *Id.* In other words, a server process provides services, and the client receives those services. A client/server relationship assumes a “clean separation of functions”—both the client and the server are independently operating programs, each performing separate functions. See, e.g., Stephen L. Montgomery, Object-Oriented Information Engineering: Analysis, Design, and

id. At oral argument, Apple “agree[d] . . . that [the analyzer server] has to be run separately from the client.” Oral Argument at 29:28; *see generally id.* 27:16–29:40.

Multiple Samsung experts testified that the Samsung software library programs “do[] not run on [their] own. [They run] as *part of the application* that is using” them. *See, e.g.*, J.A. 11591. Another Samsung expert testified that the client program “go[es] to the library” and “integrate[s]” the library program code into the application, at which point “the library code is no different than any other code in the application.” J.A. 11792.

Apple could point to no testimony where its expert stated that the library programs run separately. When asked at oral argument to point to testimony that shows that the Samsung software runs separately, Apple continually pointed to its expert’s testimony on the last day of trial that the Samsung software “has access to the code and it goes to the code where it is and *uses* it there.” J.A. 13037 (emphasis added). This testimony, though, shows the opposite of what Apple contends. It shows that the client application borrows or *uses* the library program code, not that the library program code runs separately. This is consistent with other testimony by the same Apple expert, admitting that the Samsung programs were not “standalone program[s].” J.A. 13054. As he testified, shared library code, like the Samsung software, “needs to be *exercised* by a particular application. It’s not written

Implementation 265 (1994); U.S. Patent No. 5,546,583, col. 1 ll. 24–25 (“Client/server interaction provides a clean separation of functions between processes . . .”) (filed in 1994); *see also Parallel Networks, LLC v. Abercrombie & Fitch Co.*, 704 F.3d 958, 969 (Fed. Cir. 2013) (finding that the term “generated by the server” could not cover a situation where the function was “finalized at the client”).

as a standalone program, even though it is distinct and separate from the application.” *Apple Inc. v. Samsung Elecs. Co.*, No. 5:12-cv630, ECF No. 1928 (Trial Transcript of Apr. 28, 2014), at 3052:3–6 (emphasis added). Thus, both the Samsung and Apple expert testimony showed that the shared library code is “used” by the Messenger and Browser applications, and not run separately.⁶

⁶ Further undermining Apple’s arguments that a shared library program can be a separately running server is testimony from one of the inventors of the ’647 patent taken during deposition and referenced during examination of the experts. The inventor understood that a shared library program and a server were two different ways of implementing the function described in the ’647 patent, testifying that a shared library implementation was a “different kind of implementation” than a client-server implementation. *Apple Inc. v. Samsung Elecs. Co.*, No. 5:12-cv630, ECF No. 1928 (Trial Transcript of Apr. 28, 2014), at 3045–46; *Id.*, ECF No. 1624 (Trial Transcript of April 7, 2014), at 897–99.

According to the referenced testimony, the inventor considered using a shared library to implement the functions described but opted for a server implementation instead. *Id.* Although inventor testimony “cannot be relied on to change the meaning of the claims,” *Howmedica Osteonics Corp. v. Wright Medical Technology, Inc.*, 540 F.3d 1337, 1346 (Fed. Cir. 2008) (citing *Markman v. Westview Instruments, Inc.*, 52 F.3d 967, 983 (Fed. Cir. 1995) (en banc)), “[t]he testimony of an inventor, of course, may be pertinent as a form of expert testimony, for example, as to understanding the established meaning of particular terms in the relevant art,” *Howmedica*, 540 F.3d at 1352 n.5 (citing *Phillips v. AWH Corp.*, 415 F.3d 1303, 1318 (Fed. Cir. 2005) (en banc)).

Apple emphasizes conflicting testimony between the experts for each side as to whether the Samsung software is “copied” from the library before it is run. Samsung’s expert testified that “[w]hen an application, like Messenger, uses [a shared library program], it gets [its] own copy.” J.A. 13094. Apple’s expert disagreed, stating that each application does not have its own copy of the shared library. J.A. 13036. This testimony is, indeed, conflicting and confusing.⁷ But this conflicting testimony is not relevant to whether the software on the Samsung devices runs separately or is run by the client application. Regardless of whether the code is copied, the expert testimony from both sides shows that the Samsung software library programs are not “standalone” programs that run separately.

In short, Apple provided no evidence that the accused software library programs in the Samsung phones run separately from the Browser and Messenger applications. No reasonable jury could have concluded that the accused devices had “an analyzer server for detecting structures in the data, and for linking actions to the detected structures.” We reverse the district court’s denial of JMOL of non-infringement by the Samsung devices of claim 9 of the ’647 patent.

⁷ It is unclear to what extent the experts are talking about copying the code into “Random Access Memory” (RAM) for execution, *see, e.g.*, ’647 patent, col. 3 ll. 44–46 (describing how software can be copied from disk storage to RAM prior to execution), or whether the experts are talking about making a copy from one part of disk storage to another part of disk storage. The testimony might not, in fact, be inconsistent if the experts are referring to different types of copying.

II. The Apple '721 and '172 Patents

Apple asserted claim 8 of the '721 patent and claim 18 of the '172 patent. Before trial, the district court granted Apple summary judgment of infringement of the '172 patent. The jury found both patents not invalid and found the asserted claim of the '721 patent infringed, awarding \$2,990,625 for infringement of the '721 patent by three Samsung products and \$17,943,750 for infringement of the '172 patent by seven Samsung products. Additionally, the jury found that Samsung had willfully infringed the '721 patent, which Apple argued supported an award of enhanced damages. The district court denied Samsung's motions for JMOL of invalidity and non-infringement, but granted JMOL that Samsung did not willfully infringe the '721 patent. On appeal, Samsung challenges the determination as to invalidity, and Apple challenges the JMOL as to willfulness.

We first consider the questions of patent invalidity. Samsung argues on appeal that the district court erred in not granting its motion for JMOL that the '721 and '172 patents would have been obvious in light of the various prior art references.

A patent is invalid for obviousness "if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains." 35 U.S.C. § 103(a) (pre-America Invents Act); *see also KSR Int'l Co. v. Teleflex, Inc.*, 550 U.S. 398 (2007). Obviousness is a question of law based on underlying findings of fact. *In re Kubin*, 561 F.3d 1351, 1355 (Fed. Cir. 2009). Secondary considerations, such as commercial success, long felt but unsolved needs, and the failure of others, must be considered. *In re Cyclobenzaprine Hydrochloride Extended-Release Capsule*

Patent Litig., 676 F.3d 1063, 1075 (Fed. Cir. 2012). For such evidence to be probative of nonobviousness, a patentee must demonstrate a nexus between the patented features and the particular evidence of secondary considerations. *Ashland Oil, Inc. v. Delta Resins & Refractories, Inc.*, 776 F.2d 281, 305 n.42 (Fed. Cir. 1985).

A. The Apple '721 Patent

Samsung contends that the district court should have granted its motion for JMOL that the '721 patent would have been obvious. We agree.

The '721 patent is directed to the “slide to unlock” feature of the iPhone. As described in the specification, one problem with a portable device with a touchscreen is the accidental activation of features. When a user puts the portable device in a pocket, features may be activated by unintentional contact with the screen, and, for example, a phone call might be made. Thus, cell phone manufacturers had long used “well-known” procedures to prevent this, by locking the phone (i.e., not recognizing any touch inputs) until the user has “press[ed] a predefined set of buttons . . . or enter[ed] a code or password” to “unlock” the device. '721 patent, col. 1 ll. 47–50. The '721 patent claims a particular method of unlocking. The user touches one particular place on the screen where an image appears and, while continuously touching the screen, moves his finger to move the image to another part of the screen.

Asserted claim 8 depends on claim 7. Claim 7 reads:

A portable electronic device, comprising:

- a touch-sensitive display;
- memory;
- one or more processors; and

one or more modules stored in the memory and configured for execution by the one or more processors, the one or more modules including instructions:

to detect a contact with the touch sensitive display at a first predefined location corresponding to an unlock image;

to continuously move the unlock image on the touch-sensitive display in accordance with the movement of the detected contact while continuous contact with the touch-sensitive display is maintained, wherein the unlock image is a graphical, interactive user-interface object with which a user interacts in order to unlock the device; and

to unlock the hand-held electronic device if the unlock image is moved from the first predefined location on the touch screen to a predefined unlock region on the touch-sensitive display.

'721 patent, col. 19 l. 50–col. 20 l. 9. Claim 8 additionally requires “instructions to display visual cues to communicate a direction of movement of the unlock image required to unlock the device.” *Id.* at col. 19 ll. 10–12.

At trial, Samsung presented two prior art references, the NeoNode N1 Quickstart Guide (“Neonode”) from 2004 and a video and paper by Plaisant that were presented at a computer-human-interactivity conference in 1992. The parties treat the Plaisant video and paper as a single

reference, and we do the same. Both NeoNode and Plaisant are prior art. Samsung argues that these two references together disclose every limitation of claim 8 of the '721 patent and that it would be a trivial matter for one of skill in the art to combine the teachings of these two references. Thus, it asserts that claim 8 would have been obvious because it is simply “the combination of familiar elements according to known methods.” *KSR*, 550 U.S. at 416.

The Neonode reference describes an unlocking mechanism for a touchscreen phone where a user can, through movement of a finger continuously touching the screen of the device, unlock the phone. The reference also describes text on the device indicating how the user is to unlock the phone, specifically that the user is to “Right sweep to unlock.”

KEYLOCK - UNLOCKING THE UNIT



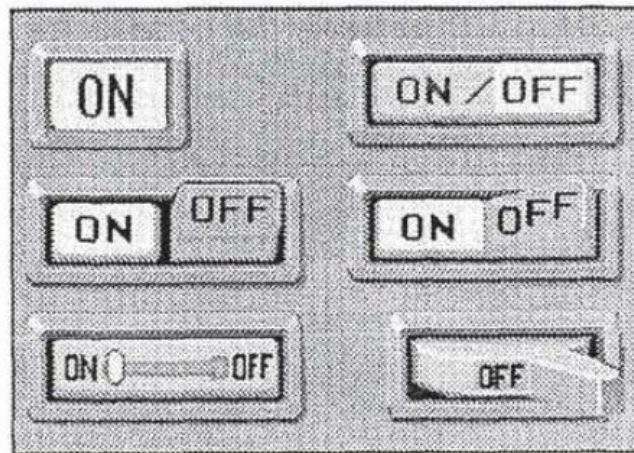
The ON/OFF switch is located on the left side of the N1, below the screen.

1. Press the power button once.
2. The text “Right sweep to unlock” appears on the screen. Sweep right to unlock your unit.

J.A. 20725. Samsung contends, and Apple does not dispute, that Neonode discloses all of the limitations of claim 8 except for limitations concerning an “unlock image” or the visual depiction of its movement. The claim requires using a “predefined location corresponding to an unlock image,” “continuous[] move[ment]” of the unlock image, and unlocking the device if the unlock image is moved from “one location to another.” In other words, Neonode discloses using a touch gesture on the screen to

unlock a phone but does not have a moving image associated with the gesture.

The Plaisant paper, Samsung argues, supplies the missing element. The Plaisant paper “compares six different touchscreen-based toggle switches to be used by novice or occasional users to control two state (on/off) devices in a touchscreen environment.” J.A. 20742. In one of these toggles, the “slider toggle,” “a sliding/dragging movement is required to change the position of the yellow pointer from one side of the toggle to the other. . . . Users can [] grab the pointer and slide it to the other side.” J.A. 20743. The “lever toggle” has the same functionality with a different appearance. These six methods are pictured below, with the “slider toggle” on the bottom left and the “lever toggle” bottom right:



J.A. 20742. As demonstrated on the video of the conference presentation, the user will place his finger at one end of the slider (the first predefined location) and will continuously move his finger to the other end of the slider (the second predefined location). While the user is moving his finger, the screen display will move the image.

On appeal, Apple does not dispute that Plaisant, when combined with Neonode, discloses all of the claimed features of the '721 patent. Rather, Apple argues that the jury could have reasonably found that (1) Plaisant teaches away from using the “slider toggle” and (2) a skilled artisan would not have had the motivation to combine Neonode and Plaisant because Plaisant describes wall-mounted devices rather than portable mobile phones.

First, Apple argues that Plaisant teaches away because the reference, in describing the results of human testing of the various slider designs, indicated that sliders were less intuitive than some other designs used. Specifically, the Plaisant paper states that “[t]he toggles that are pushed seemed to be preferred over toggles that slide. A possible explanation is that sliding is a more complex task than simply touching, but we also noticed that sliders are more difficult to implement than buttons!” J.A. 20743.

Our cases have recognized the “mere disclosure of more than one alternative” does not amount to teaching away from one of the alternatives where the reference does not “criticize, discredit, or otherwise discourage the” solution presented by the disclosure. *SightSound Techs., LLC v. Apple Inc.*, 809 F.3d 1307, 1320 (Fed. Cir. 2015) (internal quotation marks omitted) (quoting *In re Fulton*, 391 F.3d 1195, 1201 (Fed. Cir. 2004)); *Allergan, Inc. v. Apotex Inc.*, 754 F.3d 952, 963–64 (Fed. Cir. 2014). Moreover, a motivation to use the teachings of a particular prior art reference need not be supported by a finding that that feature be the “preferred, or the most desirable.” *Fulton*, 391 F.3d at 1200. Indeed, we have found a reference to not teach away when, for example, it described a particular composition “as somewhat inferior to some other product for the same use.” *In re Gurley*, 27 F.3d 551, 553 (Fed. Cir. 1994).

The fact that the Plaisant paper here notes that users did not prefer the particular design of the slider toggles is not evidence of teaching away. The reference simply discloses that users were able to figure out the push-button-type toggles more intuitively than the slider toggle. Only a single sentence in the reference suggests that sliding toggles might be less preferable to push-button-type toggles because “sliding is a more complex task than simply touching” and is “more difficult to implement.” J.A. 20743. This was so primarily because of the design of Plaisant’s sliding toggle. The Plaisant paper notes that a simple alteration of the design could solve this problem, noting that “the slider pointer should be larger, and the lever or pointer should highlight when touched to signify that the user has control over it.” *Id.* The authors also discuss positive results, noting that “[e]ven if sliders were not preferred, the fact that users used them correctly is encouraging.” *Id.* The reference also lists many benefits of sliding toggles, noting that “many other controls can be designed using sliding motions. Another advantage of the sliding movement is that it is less likely to be done inadvertently therefore making the toggle very secure. . . . This advantage can be pushed further and controls can be designed to be very secure.” *Id.* There was no criticism of sliding toggles that would lead one of skill in the art to be “discouraged from following the path” that was taken. *Gurley*, 27 F.3d at 553. Further, the reference extolls the virtues of sliding toggles as a possible solution to particular problems in computer-human-interaction design. Under our authority, a reasonable jury could not have found that Plaisant teaches away from using sliding toggles.

Apple also argues that the jury could have found that a skilled artisan would not have been motivated to combine Plaisant with Neonode because Plaisant is not relevant prior art. Whether a reference in the prior art is

“analogous” is a fact question. *In re Clay*, 966 F.2d 656, 658 (Fed. Cir. 1992). A reference qualifies as analogous prior art if it is “from the same field of endeavor, regardless of the problem addressed” or “if the reference is not within the field of the inventor’s endeavor, . . . the reference still is reasonably pertinent to the particular problem with which the inventor is involved.” *Wyers v. Master Lock Co.*, 616 F.3d 1231, 1237 (Fed. Cir. 2010) (quoting *Comaper Corp. v. Antec, Inc.*, 596 F.3d 1343, 1351 (Fed. Cir. 2010)). We conclude that no reasonable jury could find that the Plaisant reference is not analogous art in the same field of endeavor as the ’721 patent. The field of endeavor is determined “by reference to explanations of the invention’s subject matter in the patent application, including the embodiments, function, and structure of the claimed invention.” *In re Bigio*, 381 F.3d 1320, 1325 (Fed. Cir. 2004); *see also In re Deminski*, 796 F.2d 436, 442 (Fed. Cir. 1986) (finding that if a prior art reference discloses essentially the same structure and function as the invention, it is likely in the same field of endeavor).

Samsung presented expert testimony that a person of skill in the art “would be highly interested” in both Neonode and Plaisant when faced with the inadvertent activation problem because “they both deal with touch base[d] systems, they both deal with user interfaces. They both talk about changing state. . . . [A] person looking at this would just think it natural to combine these two.” J.A. 11982. Notably, Apple did not offer any expert testimony that Plaisant was not relevant to the subject matter of the ’721 patent but instead simply asserts that “Plaisant describes a wall-mounted device to control home appliances like air-conditioning units and heaters, which a skilled artisan would not naturally turn to for solving the ‘pocket dialing’ problem.” Br. for Resp’ts 26–27.

Neither the Plaisant reference nor the '721 patent so strictly defines the field of endeavor. As is described in the patent itself, the invention of the '721 patent “relate[s] generally to user interfaces that employ touch-sensitive displays, and more particularly, to the unlocking of user interfaces on portable electronic devices.” ’721 patent, col. 1 ll. 18–21. The purpose of the invention is to allow “more efficient, user-friendly procedures for transitioning such devices, touch screens, and/or applications between user interface states (e.g., from a user interface state for a first application to a user interface state for a second application, between user interface states in the same application, or between locked and unlocked states).” *Id.* at col. 1 ll. 58–67. Accordingly, the patentee included as potentially relevant many prior art references relating generally to human-interface design, including the Plaisant reference.⁸ See File Wrapper for ’721 patent, Information Disclosure Statement filed May 13, 2011. The specification clearly describes the field of the invention as being related to “transitioning” touch screen devices between interface states. ’721 patent, col. 1 ll. 58–64. The Plaisant paper describes exactly this same function—it describes “toggle switches^[9] to be used by novice or occasional users to control two state (on/off) devices in a

⁸ We have held that submission of an information disclosure statement to the USPTO does not constitute an admission that the reference listed is material prior art. *Abbott Labs. v. Baxter Pharm. Prods., Inc.*, 334 F.3d 1274, 1279 (Fed. Cir. 2003) (finding that of listing a prior sale in an IDS was not a disclaimer of claim scope). However, the nature of the prior art listed in an information disclosure statement can be informative as to the field of endeavor.

⁹ Toggle switches in Plaisant include the “sliding toggles” that are pertinent here.

touchscreen environment.” J.A. 20742 (footnote not in original). Though the authors of Plaisant describe one “practical orientation” of their work as being related to integrated control systems for entertainment, security, and climate controls, the goal of the study “was to select a usability-tested/error-free toggle and to better understand some of the problems and issues involved in the design of controls for a touchscreen environment” more broadly. *Id.*

Both the ’721 patent and the Plaisant reference also disclose essentially the same structure—a touchscreen device with software that allows the user to slide his finger across the screen to change interface states. Certainly, the problem faced by both the inventors of the ’721 patent and the authors of Plaisant was similar—how to create intuitive, easy to understand interfaces for changing states on touchscreen devices. A skilled artisan would naturally turn to references like Plaisant to find solutions. *See Bigio*, 381 F.3d at 1327 (a toothbrush was relevant prior art for a hairbrush because of the similarity in structure between the two devices); *Automatic Arc Welding Co. v. A.O. Smith Corp.*, 60 F.2d 740, 743–44, 745 (7th Cir. 1932) (an electric arc lamp was analogous art to a patent on an electric arc welder because “the problem of the electrical engineer in the other fields was so similar, and necessarily so, that one trained as an electrical engineer must be chargeable with knowledge common to those who labored in those fields”). A reasonable jury could not conclude otherwise.

Apple argues that even if Samsung established a prima facie case of obviousness, the evidence of secondary considerations demonstrates nonobviousness. Certainly secondary considerations “must be considered in evaluating the obviousness of a claimed invention.” *Transocean Offshore Deepwater Drilling, Inc. v. Maersk Contractors USA, Inc.*, 617 F.3d 1296, 1305 (Fed. Cir. 2010). But “weak secondary considerations generally do not overcome

a strong prima facie case of obviousness.” *W. Union Co. v. MoneyGram Payment Sys., Inc.*, 626 F.3d 1361, 1373 (Fed. Cir. 2010) (citations omitted); *see also Tokai Corp. v. Easton Enters., Inc.*, 632 F.3d 1358, 1371 (Fed. Cir. 2011) (“A strong case of *prima facie* obviousness . . . cannot be overcome by a far weaker showing of objective indicia of nonobviousness.”); *Leapfrog Enters., Inc. v. Fisher-Price, Inc.*, 485 F.3d 1157, 1162 (Fed. Cir. 2007) (finding that even “substantial evidence of commercial success, praise, and long felt need” was “inadequate” to overcome a strong prima facie showing of obviousness). This is particularly true when an invention involves nothing more than “the predictable use of prior art elements according to their established functions.” *Wyers*, 616 F.3d at 1246 (quoting *KSR*, 550 U.S. at 417); *see also Ohio Willow Wood Co. v. Alps South, LLC*, 735 F.3d 1333, 1344 (Fed. Cir. 2013) (“[W]here a claimed invention represents no more than the predictable use of prior art elements according to established functions, . . . evidence of secondary indicia are frequently deemed inadequate to establish non-obviousness.”).

Here, the prima facie case of obviousness was strong. Apple’s evidence of secondary considerations was weak and did not support a conclusion that the ’721 patent was nonobvious.

Apple contends that there was evidence showing (1) a long-felt but unresolved need, (2) industry praise, (3) copying, and (4) commercial success.

For long-felt but unresolved need, Apple argues that “[b]efore Apple’s invention, phone designers tried for years to solve the accidental activation problem and only came up with ‘frustrating’ methods.” Br. for Resp’ts 28. For this, it points to testimony by one of its expert witnesses describing the problem that the ’721 patent was meant to solve. After describing the “pocket dial” problem

(i.e., the accidental activation of features on touch screen phones), the expert described an example of how another manufacturer had solved the problem—the unlocking mechanism of a Nokia device. J.A. 10638–39. The expert testified that the Nokia device “shows *an example* that *I* have been very frustrated with” because “[w]hat was required to unlock, it was entirely unintuitive.” J.A. 10638 (emphasis added). What that device lacked, apparently, was a more intuitive unlocking mechanism.

We have held that evidence of a long-existing need in the industry for the solution to a recognized and persistent problem may lend support to a conclusion that an invention was nonobvious. *See, e.g., Ecolchem, Inc. v. S. California Edison Co.*, 227 F.3d 1361, 1376 (Fed. Cir. 2000). The idea behind this secondary consideration is that if a particular problem is identified by an industry but left unsolved, the failure to solve the problem (despite the incentive to do so) supports a conclusion of nonobviousness. *See, e.g., Natalie A. Thomas, Secondary Considerations in Nonobviousness Analysis: The Use of Objective Indicia Following KSR v. Teleflex*, 86 N.Y.U. L. Rev. 2070, 2078 (2011). Thus, to demonstrate long felt need, the patentee must point to an “*articulated identified* problem and evidence of efforts to solve that problem” which were, before the invention, unsuccessful. *Tex. Instruments v. Int’l Trade Comm’n*, 988 F.2d 1165, 1178 (Fed. Cir. 1993) (emphasis added). But “[w]here the differences between the prior art and the claimed invention are . . . minimal . . . it cannot be said that any long-felt need was unsolved.” *Geo. M. Martin Co. v. Alliance Mach. Sys. Int’l, LLC*, 618 F.3d 1294, 1304 (Fed. Cir. 2010).

Apple appears to identify the unsolved problem as the lack of an “intuitive” method of unlocking a touch-screen portable device. But Apple provided no evidence showing that this problem was recognized in the industry. No

reasonable jury could find testimony by a single expert about his personal experience with one device as evidence of an industry-wide long-felt need. Apple's contention here is nothing more than an unsupported assertion that Apple's method is better and more "intuitive" than previous methods. This is not sufficient to demonstrate the existence of a long-felt but unmet need. *See Perfect Web Techs., Inc. v. InfoUSA, Inc.*, 587 F.3d 1324, 1332–33 (Fed. Cir. 2009) (finding that patentee failed to demonstrate, as a matter of law, a long-felt but unmet need with bare assertions that the patent provided "improved efficiency").

As evidence of industry praise, Apple presented expert testimony that the attendees at an Apple event manifested approval when Steve Jobs first presented and unlocked the iPhone. We have held that "[a]ppreciation by contemporaries skilled in the field of the invention is a useful indicator of whether the invention would have been obvious to such persons at the time it was made." *Vulcan Eng'g Co., Inc. v. Fata Aluminium, Inc.*, 278 F.3d 1366, 1373 (Fed. Cir. 2002) (citing *Stratoflex, Inc. v. Aeroquip Corp.*, 713 F.2d 1530, 1538 (Fed. Cir. 1983)). For example, expression of disbelief by experts and then later acquiescence to the invention may be strong evidence of nonobviousness. *See, e.g., United States v. Adams*, 383 U.S. 39, 52 (1966); *Envtl. Designs, Ltd. v. Union Oil Co. of Cal.*, 713 F.2d 693, 697–98 (Fed. Cir. 1983). Similarly, industry recognition of the achievement of the invention, such as awards, may suggest nonobviousness provided that the praise is tied to the invention claimed by the patent. *See Muniauction, Inc. v. Thomson Corp.*, 532 F.3d 1318, 1327 (Fed. Cir. 2008). Evidence of approval by Apple fans—who may or may not have been skilled in the

art—during the presentation of the iPhone is not legally sufficient.¹⁰

As to copying, Apple also argues that internal Samsung documents show that a feature of the Samsung unlock mechanism was copied from the iPhone. These documents show that Samsung engineers recommended modifying Samsung software to “clarify the unlocking standard by sliding” to make it the “[s]ame as [the] iPhone.” J.A. 51289. What was copied was not the iPhone unlock mechanism in its entirety, but only using a fixed starting and ending point for the slide, a feature shown in the Plaisant prior art.

We have found, “[i]n some cases, evidence that a competitor has copied a product embodying a patented invention can be an indication of nonobviousness.” *W.M. Wrigley Jr. Co. v. Cadbury Adams USA LLC*, 683 F.3d 1356, 1364 (Fed. Cir. 2012). Evidence of copying of a feature in a patent owner’s commercial product is “not sufficient to demonstrate nonobviousness of the claimed invention” where, as here, there is a “substantial question of validity raised by the prior art references” cited by the accused infringer. *Amazon.com, Inc. v. Barnesandnoble.com, Inc.*, 239 F.3d 1343, 1366 (Fed. Cir. 2001). Thus Apple’s evidence showing that Samsung copied one aspect

¹⁰ Apple also relies on statements from Samsung documents that it contends demonstrates a competitor’s praise. We have sometimes recognized that, a competitor’s public statements, such as in advertising, touting the benefits of the technology claimed by a patent may be “inconsistent” with a position that the claimed invention is obvious. *Power-One, Inc. v. Artesyn Techs., Inc.*, 599 F.3d 1343, 1352 (Fed. Cir. 2010). These internal Samsung documents are not such public statements.

of the Apple unlocking mechanism is entitled to little weight on the question of obviousness.

Lastly, Apple points to the commercial success of the iPhone as evidence of nonobviousness. Apple argues that the success of the iPhone is tied to the patented feature of claim 8 of the '721 patent. To make this connection, Apple cites to a study where users were asked to assess their willingness to purchase a product with and without the slide-to-unlock feature. But this study only asked about tablet devices with a screen size larger than seven inches, not phones. Further, evidence that customers prefer to purchase a device "with" a slide-to-unlock capacity does not show a nexus when the evidence does not show what alternative device consumers were comparing that device to. For example, it is not clear whether the alternative device had any unlocking feature. A reasonable jury could therefore not find a nexus between the patented feature and the commercial success of the iPhone.

In short, Apple's evidence of secondary considerations is "insufficient as a matter of law to overcome our conclusion that the evidence *only* supports a legal conclusion that [the asserted claim] would have been obvious." *DyStar Textilfarben GmbH & Co. Deutschland KG v. C.H. Patrick Co.*, 464 F.3d 1356, 1371 (Fed. Cir. 2006). We reverse the judgment of infringement and no invalidity because the asserted claim of the '721 patent would have been obvious in light of Neonode and Plaisant.

B. The Apple '172 Patent

Samsung also contends that the district court erred in denying its motion for JMOL that asserted claim 18 of the '172 patent was obvious. Again, we agree.

The '172 patent covers the iPhone's "autocorrect" feature. As is described in the patent specification, the small

size of a physical or virtual keyboard on portable devices leads to more “typing mistakes and thus more backtracking to correct the mistakes. This makes the process of inputting text on the devices inefficient and reduces user satisfaction with such portable devices.” ’172 patent, col. 1 ll. 31–35. The ’172 patent seeks to solve this problem by providing methods of automatically correcting typographical errors as the user is typing. Apple asserted claim 18 of the ’172 patent, which reads:

A graphical user interface on a portable electronic device with a keyboard and a touch screen display, comprising:

- a first area of the touch screen display that displays a current character string being input by a user with the keyboard; and

- a second area of the touch screen display separate from the first area that displays the current character string or a portion thereof and a suggested replacement character string for the current character string;

wherein;

- the current character string in the first area is replaced with the suggested replacement character string if the user activates a key on the keyboard associated with a delimiter;

- the current character string in the first area is replaced with the suggested replacement character string if the user performs a gesture on the suggested replacement character string in the second area; and

the current character string in the first area is kept if the user performs a gesture in the second area on the current character string or the portion thereof displayed in the second area.

'172 patent, col. 12 l. 49–col. 13 l. 4. In essence the claim requires that current text be displayed in a first area, that the current word as typed and suggested corrections be displayed in a second area, and that the correction be automatically entered if a certain key, such as the space bar, is pressed or if the user touches the suggested replacement. Additionally, the user can choose to use the current word (as typed) if he touches that option in the second area. Figure 4D from the '172 patent specification below demonstrates the invention:

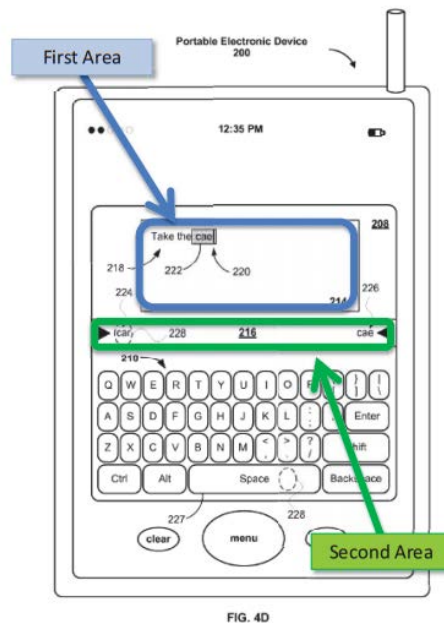


FIG. 4D

J.A. 50822 (annotations added).

There is no dispute that autocorrection features were known in the prior art. Samsung presented two pieces of

prior art that it contends together teach every limitation of the claimed invention. The first is U.S. Patent No. 7,880,730 to Robinson (“Robinson”). Robinson is directed to a “keyboard system with automatic correction” which describes a touchscreen keyboard that can automatically correct incorrectly typed text. J.A. 20885. In this invention, a pop-up window appears as a user is typing a word, displaying the current character string and a list of suggested replacements, as demonstrated in Figure 1B of the Robinson patent:

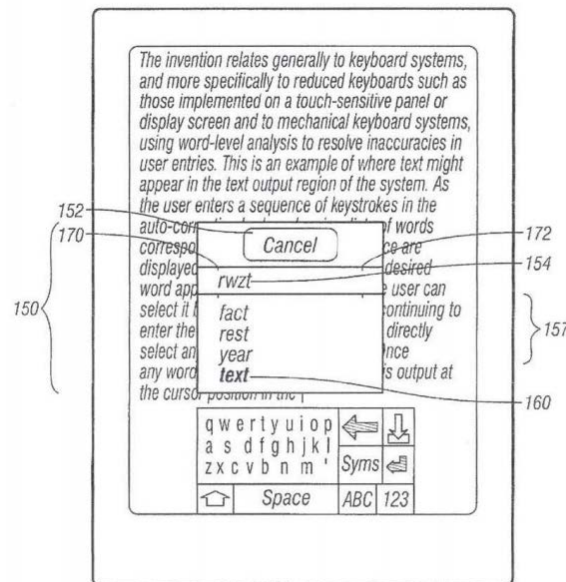


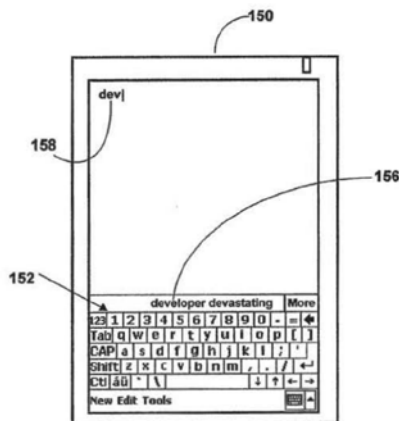
FIG. 1B

J.A. 20890.

The pop-up menu of Robinson (150) includes the word as typed (154) and suggestions, including the most commonly used suggested replacement (160), corresponding to the “second area” of claim 18 of the ’172 patent. As to the other elements, Robinson states that “[t]he space key acts

to accept the default word . . . and enters the [default] word [] in the text output region at the insertion point in the text being generated where the cursor was last positioned.” J.A. 20925 col. 33 ll. 12–16. In other words, in Robinson, pressing the space bar selects the most frequently used word that is a correction of the incorrectly typed text. Robinson also discloses that when a user selects a corrected word by touching it, or when a user selects the text as typed by touching it, the selected text will be inserted. As both parties agree, Robinson thus discloses every aspect of the invention except displaying and replacing an incorrectly typed word in a first area (in context).

Samsung argues that “displaying what a user is typing (i.e., the current character string) in the text entry area was a well-known behavior in computers.” Pet’r’s Br. 43. It points to an International Patent Application, WO 2005/008899 A1 (“Xrgomics”), which describes another text-entry system. Xrgomics discloses a “letter and word choice text input method” and describes “quick selection of choices to be implemented seamlessly for reduced keyboard systems,” like those in mobile devices. J.A. 21002. As pictured below, Xrgomics teaches displaying the current character string in a first area (158) and potential completions and/or replacements in a second area (156):



J.A. 21049. The combination of Robinson and Xrgomics results in Apple's invention.

Apple argues that the jury could have found that a skilled artisan would not have been motivated to combine Xrgomics with Robinson because Xrgomics primarily discloses a text completion (rather than text correction) system and that this is a different field than an autocorrect system. But, as with the '721 patent, the specification does not so narrowly draw boundaries around the field of the invention, stating that the disclosed invention "relate[s] generally to text input on portable electronic devices." '172 patent, col. 1 ll. 15–16. Both the '172 patent and Xrgomics disclose text input systems on a mobile device, and do so with remarkably similar structures (displaying typed text in context and corrections/completions in a space below). Considering the "reality of the circumstances—in other words, common sense," a skilled artisan would have considered Xrgomics to be within the scope of the art searched. *In re Oetiker*, 977 F.2d 1443, 1447 (Fed. Cir. 1992). Certainly text correction and text completion are closely related problems in the "same field of endeavor" such that they would be considered analogous arts. *See, e.g., Verizon Servs. Corp. v. Cox Fibernet Va., Inc.*, 602 F.3d 1325, 1338 (Fed. Cir. 2010) (finding that references relating to telephony and wireless communication were relevant to the Internet and network protocols because the "problem facing the inventors of the Network Patents was related to" the problem faced by the prior art references). There is a strong prima facie case of obviousness.

Apple also argues that a jury could have found its evidence of secondary considerations sufficient to demonstrate nonobviousness. As to the '172 patent, Apple relies only on copying and commercial success.

For copying, Apple again points to internal Samsung documents showing that one feature of the iPhone was copied. Prior to the copying, the Samsung phones automatically corrected the typed text as the user typed. *See* J.A. 51488. On the iPhone, the correction was made only after the user “accepts or hits space.” *Id.* This feature is exactly what was disclosed in Robinson. When the feature that is copied is present in the prior art, that copying is not relevant to prove nonobviousness. *See Amazon.com*, 239 F.3d at 1366; *Wm. Wrigley Jr. Co.*, 683 F.3d at 1363; *see also In re Huai-Hung Kao*, 639 F.3d 1057, 1068 (Fed. Cir. 2011) (“Where the offered secondary consideration actually results from something other than what is both claimed and *novel* in the claim, there is no nexus to the merits of the claimed invention.”).

For commercial success, Apple again relies on survey evidence to link the commercial success of its iPhone to the subject matter of claim 18. Here, the survey does address consumer preferences for this feature on phones. Users were asked whether they would be more or less likely to purchase a smartphone at a particular price point with or without autocorrection. The survey evidence indicates that consumers were more likely to purchase smartphones with automatic correction than without automatic correction. However, the survey evidence does not demonstrate whether a consumer would be more or less likely to buy a device with the specific combination of features reflected in claim 18 of the ’172 patent as opposed to, for example, the Robinson prior art.

To be relevant, commercial success must be linked to the “merits of the claimed invention,” *Wyers*, 616 F.3d at 1246 (alterations omitted), rather than features known in the prior art. *See also Ethicon Endo-Surgery, Inc. v. Covidien LP*, No. 2014-1771, 2016 WL 145576, at *9 (Fed. Cir. Jan. 13, 2016); *Pregis Corp. v. Kappos*, 700 F.3d 1348, 1356 (Fed. Cir. 2012); *Ormco Corp. v. Align Tech., Inc.*,

463 F.3d 1299, 1312 (Fed. Cir. 2006). Apple's evidence shows that phones with autocorrection may sell better than phones without autocorrection, but it does not show that phones with the specific implementation of autocorrection embodied by claim 18 sell better than phones with other methods of autocorrection disclosed by the prior art. "A nexus must be established between the merits of the claimed invention and the evidence of commercial success before that issue becomes relevant to the issue of obviousness." *Vandenberg v. Dairy Equip. Co.*, 740 F.2d 1560, 1567 (Fed. Cir. 1984). Apple presented no evidence demonstrating a nexus between the commercial success of the iPhone and the features claimed by the patent, and accordingly the claimed evidence of commercial success is entitled to no weight.

In short, we find that Samsung presented a strong case of obviousness, showing that every element of claim 18 was present in the prior art. Apple's evidence of secondary considerations was very weak. Claim 18 of the '172 patent would have been obvious to one of skill in the art as a matter of law. Therefore, we reverse the judgment of infringement and no invalidity.

Because we have found that the asserted claims of the '721 and the '172 patents would have been obvious, we need not address Apple's argument that the jury's finding of willful infringement of the '721 patent should be reinstated nor Samsung's argument that the district court erred in construing "keyboard" in the '172 patent for purposes of determining infringement.

III. The Apple '959 Patent

Next, we turn to Apple's '959 patent. The jury found the asserted claim not invalid but not infringed. After trial, both sides filed motions for JMOL, with Samsung arguing invalidity (anticipation and indefiniteness) and

Apple arguing infringement, both of which the district court denied. Both sides appeal.

We first address the issue of infringement. The '959 patent covers “universal search” on the iPhone. In short, the patent describes a method of providing “convenient access to items of information . . . by means of a unitary interface which is capable of accessing information in a variety of locations,” such as information stored on the phone and information stored on the Internet. '959 patent, col. 2 ll. 16–20. A user will input a search term into the search bar, and the phone will search a plurality of locations, including the address book, the calendar, and the Internet. The phone then displays results from all of these various searches in a list. Apple asserted claim 25, which depends on claim 24. Claim 24 reads:

A computer readable medium for locating information from a plurality of locations containing program instructions to:

receive an information identifier;

provide said information identifier to a plurality of heuristics to locate information in the plurality of locations which include the Internet and local storage media;

determine at least one candidate item of information based upon the plurality of heuristics; and

display a representation of said candidate item of information.

Id. at col. 9 ll. 16–26. Claim 25 adds an additional limitation, “wherein the information identifier is applied separately to each heuristic.” *Id.* at ll. 27–30.

On appeal, the only issue of contention is whether the search feature on the Samsung phones “provide[s] said information identifier to a plurality of heuristics to locate information in the plurality of locations which include the Internet and local storage media,” *id.* at col. 9 ll. 20–22, specifically whether the search function on the Samsung phones “locates” information on the Internet.

The district court found that “Samsung presented sufficient rebuttal evidence to permit the jury to decide that the accused devices lack instructions to search ‘a plurality of locations which include the Internet,’ as claim 25 requires.” J.A. 103. The district court pointed to two Samsung witnesses who testified that the Samsung search function “does not search the Internet, but rather ‘blends’ data previously retrieved from a Google server and a local database.” J.A. 103–04. In other words, these experts testified that because the search function only searched information previously pulled from the Internet, it was not searching the Internet, as required by the claim language. As the district court found, this is substantial evidence supporting the jury verdict of non-infringement.

Apple argues that the plain meaning of the claim ought to cover searching information previously downloaded from the Internet. The district court found that this argument attempts to assert “a new claim construction position after trial, when Apple did not request additional claim construction, and plain and ordinary meaning applied to the terms that Apple now raises.” J.A. 104. We agree with the district court and affirm the denial of Apple’s motion for JMOL of infringement of claim 25 of the ’959 patent. We thus also affirm the judgment of non-infringement.

Samsung conceded at oral argument in our court that we need not address its appeal as to invalidity of the ’959 patent if we uphold the jury’s non-infringement finding.

Since we sustain the jury's verdict of non-infringement, we need not address issues of invalidity.

IV. The Apple '414 Patent

We now consider Apple's '414 patent. The jury found the asserted claim of the '414 patent not invalid and not infringed. After trial, both sides challenged the jury verdict, with Samsung moving for JMOL of invalidity and Apple moving for JMOL of infringement. The district court denied both motions. Both parties appeal.

We address first the issue of infringement. The '414 patent covers "background sync" and describes systems, methods, and computer readable media for synchronizing data between multiple devices. Specifically, the patent covers simultaneous synchronization where the "synchronization tasks and non-synchronization tasks [are] executed concurrently." '414 patent, col 2 ll. 19–21. Basically, this means that a user can continue using a program that manipulates data (say the Address Book) and the system can synchronize the data being used (i.e., the contacts in the Address Book) at the same time. The invention will "synchronize" a contact created on an iPhone to another device, such as an iPad, without any user interaction. Apple asserted claim 20, which depends on claim 11. Claim 11 reads:

A computer readable storage medium containing executable program instructions which when executed cause a data processing system to perform a method comprising:

executing at least one user-level non-synchronization processing thread, wherein the at least one user-level non-synchronization processing thread is provided by a user application which provides a user interface to allow a user to access

and edit structured data in a first store associated with a first database; and

executing at least one synchronization processing thread concurrently with the executing of the at least one user-level non-synchronization processing thread, wherein the at least one synchronization processing thread is provided by a synchronization software component which is configured to synchronize the structured data from the first database with the structured data from a second database.

Id. at col. 33 ll. 37–54. Claim 20 adds the additional limitation, “wherein the synchronization software component is configured to synchronize structured data of a first data class and other synchronization software components are configured to synchronize structured data of other corresponding data classes.” *Id.* at col. 34, ll. 18–22.

Apple contends that the jury’s finding of non-infringement is not supported by substantial evidence, and that the district court erred in concluding otherwise. As the district court found, “[i]t is undisputed that claim 20 requires at least three distinct ‘synchronization software components The first is the claimed synchronization software component ‘configured to synchronize structured data of a first data class’ and the other two are the ‘other synchronization software components’ configured ‘to synchronize structured data of other corresponding data classes.’” J.A. 99. In other words, the claim requires three pieces of software that will synchronize three different data classes, such as contacts, calendar, and email. It is also undisputed that the accused Samsung phones contain synchronization software components that meet the other limitations of the claims for two data classes (calendar and contacts). The only issue is

whether the Samsung devices contain synchronization software components “configured to synchronize” for email. The limitation in question was construed by the district court to have its plain and ordinary meaning.

The district court concluded that “substantial trial evidence permitted a reasonable jury to determine non-infringement” on the basis of Samsung expert testimony that email software was not configured to synchronize because it does not synchronize data by itself, but rather “indirectly ‘cause[s]’ synchronization by calling other software components.” J.A. 100; *see also, e.g.*, J.A. 11573. We agree with the district court that this is substantial evidence supporting the jury verdict of non-infringement.

Apple now argues that this testimony is insufficient because the plain and ordinary meaning of “configured to synchronize” includes indirect causes of synchronization, like the Samsung email software. The Samsung expert testimony, according to Apple, does not suffice as substantial evidence because it “‘import[s] additional limitations into the claims’ by suggesting that . . . a sync adapter be configured to perform all synchronization or to perform synchronization in a specific way.” J.A. 100. The district court rejected this argument because “Apple seeks a post-trial construction for ‘configured to synchronize’ . . . despite never requesting such a construction before.” *Id.* at 101–02. We agree and affirm the judgment of non-infringement.

Since we conclude that substantial evidence supports the jury’s finding of non-infringement, we need not address the invalidity of claim 20 of the ’414 patent.

V. The Samsung ’239 Patent

The jury, based on the district court’s claim construction, found asserted claim 15 of the ’239 patent not in-

fringed. Samsung argues that the district court erred in construing “means for transmission” in claim 15.

Samsung’s ’239 patent pertains to “remote video transmission” and “provide[s] a method and means for capturing full-color, full-motion audio/video signals, digitizing and compressing the signals into a digitized data file, and transmitting the signals over telephone lines, cellular, radio and other telemetric frequencies.” ’239 patent, col. 2 ll. 26–31. Samsung asserted claim 15, which reads:

An apparatus for transmission of data, comprising:

a computer including a video capture module to capture and compress video in real time;

means for transmission of said captured video over a cellular frequency.

Id. at col. 14 ll. 17–21. The district court construed “means for transmission”—a means-plus-function claim limitation—to require software “performing a software sequence of initializing one or more communications ports on said apparatus, obtaining a cellular connection, obtaining said captured video, and transmitting said captured video” disclosed in the specification, in addition to hardware. J.A. 150.

Samsung argues that “[t]he specification of the ’239 patent does not require any *software* for transmission, and including such software [in addition to hardware] as necessary structure was error.” Pet’r’s Br. 57 (emphasis in original). But, as the district court found, “the term ‘transmission’ implies communication from one unit to another, and the specification explains that software is necessary to enable such communication.” J.A. 144. Consistent with this, “the specification teaches that a

software sequence is necessary for transmitting a signal in the context of the invention. . . . Under the preferred embodiment, the '239 patent discloses that software is required for transmission: "Transfer software *sequence B enables the remote unit* to communicate" and "contains all of the instructions *necessary*" for communication." *Id.* (citing and quoting from the '239 patent, col. 8 ll. 23–30). Hardware, alone, does nothing without software instructions telling it what to do, and the patent recognizes this, stating that the "transfer software" is what "enables" the transmission. *See* '239 patent, col. 8 ll. 23–30. Thus, because "corresponding structure must include all structure that actually performs the recited function," *Cardiac Pacemakers, Inc. v. St. Jude Medical, Inc.*, 296 F.3d 1106, 1119 (Fed. Cir. 2002), the district court correctly included software as part of the corresponding structure for "means for transmission."

Samsung also argues, in the alternative, that even if software were required, the district court incorrectly required that the software initialize the communications ports, obtain a cellular connection, and obtain the captured video. But the district court was correct in this regard as well. The specification explicitly describes the initializing and obtaining aspects of the transfer software as part of the structure that enables the remote unit to transmit a video file over a cellular frequency. *See* '239 patent, col. 8 ll. 17–30 ("Transmission of a data file is accomplished by selecting the 'TRANSFER' button" which "initiates" specific software sequences (sequences B and C) described in the specification as initializing the communications port, obtaining a cellular connection, and obtaining the captured video.).

We affirm the district court's construction of "means for transmission" in claim 15 of the '239 patent and the judgment of non-infringement.

VII. The Samsung '449 Patent

Samsung asserted claim 27 of the '449 patent. The jury found that Apple had infringed and awarded \$158,400 in damages. The district court denied Apple's post-trial motion for JMOL of non-infringement. Apple challenges the district court's denial of its motion for JMOL that its products do not infringe the '449 patent.

Samsung's '449 patent is directed to camera systems for compressing/decompressing and organizing digital files, such as photos and videos. Samsung asserted claim 27, which depends on claim 25. Claim 25 reads:

A digital camera comprising:

a lens,

an imaging device which converts an optical image into an analog signal;

an A/D converter which converts said analog signal from said imaging device to a digital signal;

a compressor which compresses said digital signal outputted from said A/D converter, and generates compressed data by using a different compressing method for moving image signals and for still image signals;

a recording circuit which records compressed data, said compressed data including a moving image signal, and a still image signal;

a decompressor which decompresses said compressed data by using a different decompressing method according to whether

said recorded compressed data is a moving image signal or a still image signal;

a reproducing circuit which reproduces a moving image signal, a sound signal in synchronous to said moving image signal, and a still image signal; and

a display which displays said moving image signals and still image signals outputted from said reproducing circuit, and a list of said moving image signal and still image signal as a search mode, and a list of classifications as a classification mode;

wherein said recording circuit records each one of said plurality of image signals with classification data, and

said display lists a plurality of classifications and a number of images belonging to each classification.

'449 patent, col. 18 ll. 7–35 (emphases added). Claim 27 additionally requires the classification be “able to change by a direction of a user.” *Id.* at ll. 40–42.

There are three limitations at issue on appeal. First, Apple contends that no reasonable jury could have found that the Apple products met the “compressor” and “decompressor” limitations of the claim because these limitations require components that compress or decompress both still images and videos, and its products use separate and distinct components to compress and decompress still images and videos. But, as the district court found, Samsung presented testimony that “identified a single Apple design chip with the circuitry that performs both compressing methods.” J.A. 118. Even though this chip may contain separate components, a jury may still have reasonably concluded that the chip (not the individual

components of that chip) performs the “compressing” and “decompressing” steps and that the chip itself meets the “compressing” and “decompressing” limitations.

Second, Apple contends that no reasonable jury could have found that the Apple products met the “search mode” limitation because the Apple products do not display a “list,” as required by the claims. The Apple products contain a “Camera Roll” which displays an array of thumbnails (small previews of the image). Samsung presented expert testimony that this “Camera Roll” was a “list” under the plain and ordinary meaning of that term in the context of the ’449 patent. As the district court found, a jury could have believed this testimony and concluded that this limitation was met.

Lastly, Apple argues that its products do not have a recording circuit that “records each one of said plurality of image signals with classification data.” ’449 patent, col. 18 ll. 32–33. Apple argues that the Camera Roll on its products includes all photos and videos taken with the device so that there is no classification of the images. But again, Samsung presented testimony that the Apple products record images with classification data. Samsung’s expert testified that, for example, the Camera Roll contains “Albums” that are created automatically as well as albums that are created by the user. A jury could have reasonably believed this expert and found that Apple’s products contained “classification data.”

Therefore, we affirm both the district court’s denial of JMOL of non-infringement by Apple of claim 27 of the ’449 patent and the judgment of infringement.

VII. Remaining Issues

Because we have reversed the district court’s denial of JMOL of non-infringement of the ’647 patent and obviousness of the ’721 and ’172 patents, Samsung’s remain-

ing arguments relating to ongoing royalties and the district court's evidentiary rulings related to damages are now moot.

CONCLUSION

In conclusion, we reverse the district court's judgment of infringement of the '647 patent and the judgment of no invalidity with respect to obviousness of the '721 patent and the '172 patent. Samsung was entitled to a judgment of non-infringement of the '647 patent and a judgment of invalidity as to the '721 and '172 patents. We affirm the judgment of non-infringement of Apple's '959 patent, Apple's '414 patent, and Samsung's '239 patent and affirm the judgment of infringement of Samsung's '449 patent. In light of these holdings, we find that we need not address any of the other issues on appeal.

AFFIRMED-IN-PART, REVERSED-IN-PART

COSTS

Costs to Samsung.

TAB 2

Encyclopedia of Computer Science

FOURTH EDITION

Editors:

**Anthony Ralston
Edwin D. Reilly
David Hemmendinger**



Encyclopedia of Computer Science

Fourth Edition

Edited by Anthony Ralston, Edwin D. Reilly and David Hemmendinger

© Nature Publishing Group, 2000

All rights reserved. No reproduction, copy or transmission of this publication may be made without written permission. No part of this publication may be reproduced, stored in or introduced into a retrieval system, or transmitted, in any form, or by any means (electronic, mechanical, photocopying, recording or otherwise) without the prior written permission of the publisher unless in accordance with the provisions of the Copyright Designs and Patents Act 1988, or under the terms of any licence permitting limited copying issued by the Copyright Licensing Agency, 90 Tottenham Court Road, London W1P 9HE, UK.

Any person who does any unauthorized act in relation to this publication may be liable to criminal prosecution and civil claims for damages.

Published in the United Kingdom by
NATURE PUBLISHING GROUP, 2000
25 Eccleston Place, London, SW1W 9NF, UK
Basingstoke and Oxford
ISBN: 0-333-77879-0
Associated companies throughout the world
<http://www.nature.com>

British Library Cataloguing in Publication Data

Encyclopedia of Computer Science

1. Computer science—Encyclopedias

I. Ralston, Anthony II. Reilly, Edwin D. III. Hemmendinger, David

004/.03

Published in the United States and Canada by
GROVE'S DICTIONARIES, INC
345 Park Avenue South, New York, NY 10010-1707, USA
ISBN: 1-561-59248-X

Library of Congress Cataloging in Publication Data

A catalog for this record is available from the Library of Congress

Typeset in the United Kingdom by Aarontype Limited, Bristol.
Printed and bound in the United Kingdom by Bath Press Ltd, Bath.

CLIENT-SERVER COMPUTING

For articles on related subjects see DISTRIBUTED SYSTEMS; ELECTRONIC COMMERCE; OPERATING SYSTEMS; CONTEMPORARY ISSUES; and TCP/IP.

Introduction

Client-server computing is a distributed computing model in which *client* applications request services from *server* processes. Clients and servers typically run on different computers interconnected by a computer network. Any use of the Internet (*q.v.*), such as information retrieval (*q.v.*) from the World Wide Web (*q.v.*), is an example of client-server computing. However, the term is generally applied to systems in which an organization runs programs with multiple components distributed among computers in a network. The concept is frequently associated with *enterprise computing*, which makes the computing resources of an organization available to every part of its operation.

A *client* application is a process or program that sends messages to a server via the network. Those messages request the server to perform a specific task, such as looking up a customer record in a database or returning a portion of a file on the server's hard disk. The client manages local resources such as a display, keyboard, local disks, and other peripherals.

The *server* process or program listens for client requests that are transmitted via the network. Servers receive those requests and perform actions such as database queries and reading files. Server processes typically run on powerful PCs, workstations (*q.v.*), or mainframe (*q.v.*) computers.

An example of a client-server system is a banking application that allows a clerk to access account information on a central database server. All access is done via a PC client that provides a graphical user interface (GUI). An account number can be entered into the GUI along with how much money is to be withdrawn or deposited, respectively. The PC client validates the data provided by the clerk, transmits the data to the database server, and displays the results that are returned by the server.

The client-server model is an extension of the *object-based* (or *modular*) programming model, where large pieces of software are structured into smaller components that have well defined interfaces. This decentralized approach helps to make complex programs maintainable and extensible. Components interact by exchanging messages or by *Remote Procedure Calling* (RPC—see DISTRIBUTED SYSTEMS). The calling component becomes the client and the called component the server.

A client-server environment may use a variety of operating systems and hardware from multiple vendors; standard network protocols like TCP/IP provide compatibility. Vendor independence and freedom of choice are further advantages of the model. Inexpensive PC equipment can be interconnected with mainframe servers, for example.

Client-server systems can be scaled up in size more readily than centralized solutions since server functions can be distributed across more and more server computers as the number of clients increases. Server processes can thus run in parallel, each process serving its own set of clients. However, when there are multiple servers that update information, there must be some coordination mechanism to avoid inconsistencies.

The drawbacks of the client-server model are that security is more difficult to ensure in a distributed environment than it is in a centralized one, that the administration of distributed equipment can be much more expensive than the maintenance of a centralized system, that data distributed across servers needs to be kept consistent, and that the failure of one server can render a large client-server system unavailable. If a server fails, none of its clients can make further progress, unless the system is designed to be fault-tolerant (see FAULT-TOLERANT COMPUTING).

The computer network can also become a performance or reliability bottleneck; if the network fails, all servers become unreachable. If one client produces high network traffic then all clients may suffer from long response times.

Design Considerations

An important design consideration for large client-server systems is whether a client talks directly to the server, or whether an *intermediary process* is introduced between the client and the server. The former is a two-tier architecture, the latter is a three-tier architecture.

The *two-tier architecture* is easier to implement and is typically used in small environments (one or two servers with one or two dozens of clients). However, a two-tier architecture is less scalable than a three-tier architecture.

In the *three-tier architecture*, the intermediate process is used for decoupling clients and servers. The intermediary can *cache* frequently used server data to ensure better performance and scalability (see CACHE MEMORY). Performance can be further increased by having the intermediate process distribute client requests to several servers so that requests execute in parallel.

The intermediary can also act as a translation service by converting requests and replies from one format to another or as a security service that grants server access only to trusted clients.

Other important design considerations are:

- ◆ **Fat vs. thin client:** A client may implement anything from a simple data entry form to a complex business application. An important design consideration is how to partition application logic into client and server components. This has an impact on the scalability and maintainability of a client-server system. A “thin” client receives information in its final form from the server and does little or no data processing. A “fat” client does more processing, thereby lightening the load on the server.
- ◆ **Stateful vs. stateless:** Another design consideration is whether a server should be stateful or stateless. A *stateless* server retains no information about the data that clients are using. Client requests are fully self-contained and do not depend on the internal state of the server. The advantage of the stateless model is that it is easier to implement and that the failure of a server or client is easier to handle, as no state information about active clients is maintained. However, applications where clients need to acquire and release locks on the records stored at a database server usually require a *stateful* model, because locking information is maintained by the server for each individual client (see DATABASE CONCURRENCY CONTROL).
- ◆ **Authentication:** For security purposes servers must also address the problem of authentication (*q.v.*). In a networked environment, an unauthorized client may attempt to access sensitive data stored on a server. Authentication of clients is handled by using cryptographic techniques such as *public key encryption* (see CRYPTOGRAPHY, COMPUTERS IN) or special *authentication* (*q.v.*) servers such as in the OSF DCE system described below.

In public key encryption, the client application “signs” requests with its private cryptographic key (see DIGITAL SIGNATURE), and encrypts the data in the request with a secret session key known only to the server and to the client. On receipt of the request, the server validates the signature of the client and decrypts the request only if the client is authorized to access the server.

- ◆ **Fault tolerance:** Applications such as flight-reservation systems and real-time market data feeds must be fault-tolerant. This means that important services remain available in spite of the failure of part of the computer system on which the servers are running (high availability), and that no information

is lost or corrupted when a failure occurs (consistency). For the sake of high availability, critical servers can be replicated, which means they are provided redundantly on multiple computers. If one replica fails then the other replicas still remain accessible by the clients. To ensure consistent modification of database records stored on multiple servers, a transaction processing (TP—*q.v.*) monitor can be installed. TP monitors are intermediate processes that specialize in managing client requests across multiple servers. The TP monitor ensures that such requests happen in an “all-or-nothing” fashion and that all servers involved in such requests are left in a consistent state, in spite of failures.

Distributed Object Computing

Distributed object computing (DOC) is a generalization of the client-server model. Object-oriented modeling and programming are applied to the development of client-server systems. Objects are pieces of software that encapsulate an internal state and make it accessible through a well-defined *interface*. In DOC, the interface consists of object operations and attributes that are remotely accessible. Client applications may connect to a remote instance of the interface with the help of a naming service. Finally, the clients invoke the operations on the remote object. The remote object thus acts as a server.

This use of objects naturally accommodates heterogeneity and autonomy. It supports heterogeneity since requests sent to server objects depend only on their interfaces and not on their internals. It permits autonomy because object implementations can change transparently, provided they maintain their interfaces.

If complex client-server systems are to be assembled out of objects, then objects must be compatible. Client-server objects have to interact with each other even if they are written in different programming languages and run on different hardware and operating system platforms.

Standards are required for objects to interoperate in heterogeneous environments. One of the widely adopted vendor-independent DOC standards is the OMG (Object Management Group) CORBA (Common Object Request Broker Architecture) specification. CORBA consists of the following building blocks:

- ◆ **Interface Definition Language:** Object interfaces are described in a language called IDL (Interface Definition Language). IDL is a purely declarative language resembling C++. It provides the notion of interfaces (similar to classes), of interface inheritance, of operations with input and output arguments, and of data types (*q.v.*) that can be

passed along with an operation. IDL serves for declaring remotely accessible server objects in a platform- and programming language-neutral manner, but not for implementing those objects. CORBA objects are implemented in widely used languages such as C++, C, Java, and Smalltalk.

- ◆ **Object Request Broker:** The purpose of the ORB (Object Request Broker) is to find the server object for a client request, to prepare the object to receive the request, to transmit the request from the client to the server object, and to return output arguments back to the client application. The ORB mainly provides an object-oriented RPC facility.
- ◆ **Basic Object Adapter:** The BOA (Basic Object Adapter) is the primary interface used by a server object to gain access to ORB functions. The BOA exports operations to create object references, to register and activate server objects, and to authenticate requests. An object reference is a data structure that denotes a server object in a network. A server installs its reference in a *name server* so that a client application can retrieve the reference and invoke the server. The object reference provides the same interface as the server object that it represents. Details of the underlying communication infrastructure are hidden from the client.
- ◆ **Dynamic Invocation Interface:** The DII (Dynamic Invocation Interface) defines functions for creating request messages and for delivering them to server objects. The DII is a low-level equivalent of the communication stubs (message-passing interfaces) that are generated from an IDL declaration.
- ◆ **Internet Inter-ORB Protocol:** The Internet Inter-ORB Protocol (IIOP) allows CORBA ORBs from different vendors to interoperate via a TCP/IP connection. IIOP is a simplified RPC protocol used to invoke server objects via the Internet in a portable and efficient manner.
- ◆ **Interface and Implementation Repository:** The CORBA Interface Repository is a database containing type information (interface names, interface operations, and argument types) for the interfaces available in a CORBA system. This information is used for dynamic invocation via the DII, for revision control, and so forth. The Implementation Repository provides information allowing an ORB to locate and launch server objects.

Client-Server Toolkits

A wide range of software toolkits for building client-server software is available on the market today. Client-server toolkits are also referred to as *middle-*

ware. CORBA implementations are an example of well-known client-server middleware. Other examples are OSF DCE, DCOM, message-oriented middleware, and transaction processing monitors.

- ◆ **OSF DCE:** The Open Software Foundation (OSF) Distributed Computing Environment (DCE) is a *de facto* standard for multivendor client-server systems. DCE is a collection of tools and services that help programmers in developing heterogeneous client-server applications. DCE is a large and complex software package; it mainly includes a remote procedure call facility, a naming service, a clock synchronization service, a client-server security infrastructure, and a *threads* package (see MULTITASKING).
- ◆ **DCOM:** Distributed Component Object Model (DCOM) is Microsoft's object protocol that enables *ActiveX* components to communicate with each other across a computer network. An *ActiveX* component is a remote accessible object that has a well-defined interface and is self-contained. *ActiveX* components can be embedded into Web documents, so that they download to the client automatically to execute in the client's Web browser (see WORLD WIDE WEB). DCOM provides a remote instantiation facility allowing clients to create remote server objects. It also provides a security model to let programmers restrict who may create a server object and who may invoke it. Finally, an Interface Definition Language (IDL) is provided for defining remotely accessible object interfaces and composing remote procedure calls.
- ◆ **MOM:** Message-Oriented Middleware (MOM) allows the components of a client-server system to interoperate by exchanging general purpose messages. A client application communicates with a server by placing messages into a *message queue*. The client is relieved of the tasks involved in transmitting the messages to the server reliably. After the client has placed a message into a message queue, it continues other work until the MOM informs the client that the server's reply has arrived. This kind of communication is called *asynchronous messaging*, since client and server are decoupled by message queues. MOM functions much like electronic mail, storing and forwarding messages on behalf of client and server applications. Messages may be submitted even when the receiver happens to be temporarily unavailable, and are thus inherently more flexible and fault-tolerant than RPC. Examples of MOM are IBM's MQSeries product and the OMG Event Service. Web *push technologies* such as Marimba's *Castanet* also fall into the category of message-oriented middleware.

- ◆ **Transaction Processing (TP) Monitors:** Transaction processing (*q.v.*) monitors allow a client application to perform a series of requests on multiple remote servers while preserving consistency among the servers. Such a series of requests is called a *transaction*. The TP monitor ensures that either all requests that are part of a transaction succeed, or that the servers are *rolled back* to the state they had before the unsuccessful transaction was started. A transaction fails when one of the involved computers or applications goes down, or when any of the applications decides to *abort* the transaction. TP monitors are part of client-server products such as Novell's Tuxedo and Transarc's Encina.

A TP monitor can be used within a banking system when funds are withdrawn from an account on one database server and deposited in an account on another database server. The monitor makes sure that the transaction occurs in an "all or nothing" fashion. If any of the servers fails during the transfer then the transaction is rolled back such that both accounts are in the state they were before transaction was started.

Bibliography

- 1995. Mowbray, T. J., and Zahavi, R. *The Essential CORBA*. New York: John Wiley.
- 1996. Andrade, J. M. (ed.), Dwyer, T., Felts, S., and Carges, M. *The Tuxedo System; Software for Constructing and Managing Distributed Business Applications*. Reading, MA: Addison-Wesley.
- 1997. Shan, Y.-P., Earle, R. H., and Lenzi, M. A. *Enterprise Computing With Objects: From Client/Server Environments to the Internet*. Reading, MA: Addison-Wesley.
- 1998. Orfali, R., and Harkey, D. *Client/Server Programming with Java and CORBA*, 2nd Ed. New York: John Wiley.

Websites

Client-server frequently asked questions URLs: <http://www.abs.net/~lloyd/csfaq.txt>.
 OMG CORBA documentation URL: <http://www.omg.org>.
 OSF DCE documentation URL: <http://www.rdg.opengroup.org/public/pubs/catalog/dz.htm>.
 Microsoft ActiveX and related technology URL: <http://www.microsoft.com/com>.

Silvano Maffeis

CLUSTER COMPUTING

For articles on related subjects see CLIENT-SERVER COMPUTING; COOPERATIVE COMPUTING; DATABASE MANAGEMENT SYSTEM; DISTRIBUTED SYSTEMS; MULTIPROCESSING; NETWORKS; COMPUTER; PARALLEL PROCESSING; and SUPERCOMPUTERS.

Introduction

A cluster of computers, or simply a *cluster*, is a collection of computers that are connected together and

used as a single computing resource. Clusters have been used from the dawn of electronic computing as a straightforward way to obtain greater capacity and higher reliability than a single computer can provide. Clusters can be an informal, if not anarchic, computer organization. Often they have not been built by computer manufacturers but rather assembled by customers on an *ad hoc* basis to solve a problem at hand.

The first cluster probably appeared in the late 1950s or early 1960s when some company's finance officer, realizing that payroll checks wouldn't get printed if the computer broke down, purchased a spare. Software tools for managing groups of computers and submitting batch jobs to them, such as IBM's Remote Job Entry (RJE) System, became commercially available in the mid-1970s. By the late 1970s, Tandem Computers began selling highly reliable systems that were clusters, with software to make them appear to access a single database system. However, it was not until the early 1980s that DEC (Digital Equipment Corporation—*q.v.*) coined the term *cluster* for a collection of software and hardware that made several VAX minicomputers (*q.v.*) appear to be a single time-sharing (*q.v.*) system called the VAXcluster.

With the appearance of very high performance personal workstations (*q.v.*) in the early 1990s, technical computer users began replacing expensive supercomputers with clusters of those workstations which they assembled themselves. Computer manufacturers responded with prepackaged workstation clusters, which became the standard form of supercomputers by the mid-1990s; a system of this type with special-purpose added hardware achieved the milestone of defeating the reigning human chess champion, Garry Kasparov (see COMPUTER CHESS). By 1998, even those systems were being challenged by user-constructed clusters of increasingly powerful personal computers. A very large, highly diffuse and informal cluster—using spare time on approximately 22,000 personal computers owned by volunteers, connected only occasionally though the Internet—succeeded in February 1998 in decoding a "challenge" message encrypted using the Data Encryption Standard system with a 56-bit key (see CRYPTOGRAPHY, COMPUTERS IN). The answer was found by simply trying one after another of the 63 quadrillion possible keys; success came after taking only 39 days to examine 85% of the keys. Appropriately, the decoded message read "Many hands make light work."

Individual spectacular feats such as this are not, however, the reason that computer industry analysts estimated that half of all high performance server computer systems would be clusters by the turn of the century. Clusters provide a practical means of increasing

1972. Dahl, O.-J., Dijkstra, E. W., and Hoare, C. A. R. *Structured Programming*. New York: Academic Press.
1976. *ACM Computing Surveys (Special Issue: Data-Base Management Systems)*, 8, 1 (March).
1976. Fry, J. P., and Sibley, E. H. "Evolution of Data-base Management Systems," *Computing Surveys*, 8, 1, 7-42.
1978. *Bell System Technical Journal (UNIX Time-Sharing System)*, 57, 6, Part 2 (July-August).
1978. Sammet, J. E. "Roster of Programming Languages for 1976-77," *ACM SIGPLAN Notices*, 13, 11, 56-85.
1980. Knuth, D. E. and Trabb Pardo, L. "The Early Development of Programming Languages," in *A History of Computing in the Twentieth Century* (eds. N. Metropolis, J. Howlett, and G.-C. Rota), 197-273. New York: Academic Press.
1981. Weizer, N. "A History of Operating Systems," *Datamation*, 27, 1, 119-126.
1981. Wexelblat, R. (ed.) *History of Programming Languages*. ACM Monograph Series. New York: Academic Press.
1981. Date, C. J. *An Introduction to Database Systems*, 5th Ed. Reading, MA: Addison-Wesley.
1986. Lammers, S. *Programmers at Work: Interviews*. Redmond, WA: Microsoft Press.
1988. Goldberg, A. (ed.) *A History of Personal Workstations*. Reading, MA: Addison-Wesley.
1990. Sammet, J. E. "Some Approaches to, and Illustrations of, Programming Language History," *Annals of the History of Computing*, 13, 1, 33-50.
1991. Milner, R., Tofte, M., and Harper, R. *The Definition of Standard ML*. Cambridge, MA: MIT Press.
1994. Salus, P. H. *A Quarter Century of Unix*. Reading, MA: Addison-Wesley.
1994. Stroustrup, B. *The Design and Evolution of C++*. Reading, MA: Addison-Wesley.
1996. Bergin, T. M., and Gibson, R. G. (eds.) *History of Programming Languages II*. New York: ACM Press; Reading, MA: Addison-Wesley.
1998. Ceruzzi, P. E. *A History of Modern Computing*. Cambridge, MA: MIT Press.

Jean E. Sammet, revised by Michael S. Mahoney

SOFTWARE LIBRARIES, NUMERICAL AND STATISTICAL

For articles on related subjects see COMPATIBILITY; COMPUTER ALGEBRA; DOCUMENTATION; MATHEMATICAL SOFTWARE; SOFTWARE PORTABILITY; and SUBPROGRAM.

A *program library* is a collection of computer programs for a particular application. To be characterized as a library, such a collection should contain a substantial number of computer program modules designed to solve a wide range of problems in the given area. In addition, the programs in a library should be coherent, both in their external appearance and their internal design. In particular, they should

- ◆ present a similar user interface (*q.v.*),
- ◆ have a fixed documentation format,
- ◆ be designed to be used easily in combination,
- ◆ be built upon a common set of low-level utilities,
- ◆ share coding and portability standards.

This article describes program libraries for general-purpose numerical computation and statistical analysis. This includes, for example, the evaluation of the special functions of applied mathematics, the numerical solution of differential equations, regression, and analysis of variance. Most scientific computing facilities make libraries of this type available to their users.

Program Library Development

The first program libraries for numerical computation were written in machine or assembly language for a particular computer at a given site. Probably the earliest of these was a library written for the EDSAC (*q.v.*) computer in England by Wilkes (*q.v.*), Wheeler, and Gill in 1951. By the early 1960s computer manufacturers were working on program libraries to help customers and stimulate sales, and in 1961 IBM released the SSP (Scientific Subroutine Package) library, providing it free with a computer rental or sale.

At the same time, many groups in universities, government laboratories, and private industry began to feel the need to consolidate programming effort into useful libraries. For example, statisticians in the biomedical group at the University of California put together a group of statistical routines known as the BMDP library, the first edition appearing in 1961. Other statistical libraries originating in the early 1960s were SPSS (Statistical Package for the Social Sciences), originally written at Stanford University and further developed by the National Opinion Research Center at the University of Chicago, and SAS (Statistical Analysis System), developed at North Carolina State. Each of these is now supported commercially. By the late 1960s software library development was also occurring in the private sector, e.g. at Boeing and Monsanto, although the resulting libraries were primarily for internal use.

Libraries for numerical computation were also being built in England during this period. One was developed in 1963 for the IBM Stretch (*q.v.*) computer at the Harwell Atomic Energy Research Establishment; in 1967 the library was converted to the IBM 360; its successor remains available today. In 1970, six British computing centers began an effort to develop a library for their ICL 1906A/S computers, and in 1971 Mark 1 of the NAG (Nottingham Algorithms Group) library was released. Implementation for other computer systems followed, and, by 1976, a non-profit company, Numerical Algorithms Group Ltd, had been formed to continue development and distribution. The NAG effort continues to be characterized by close collaboration between a full-time coordination staff and a large number of specialists in numerical and statistical analysis in university and government research institutions worldwide. NAG now markets general-purpose

numerical libraries in Fortran (*q.v.*), C (*q.v.*), and Ada (*q.v.*), as well as specialized libraries for parallel computing and other mathematical topics.

Probably the first commercial venture formed exclusively to market a general-purpose mathematical subroutine library was IMSL (International Mathematical and Statistical Libraries), which was incorporated in 1970. The next year IMSL released a library for the IBM 360/370 class of computers and sold seven copies. By 1976, when the company showed its first profit, it had 430 customers using its library on seven different computer systems. Today the IMSL Libraries are developed and marketed by Visual Numerics, Inc., which offers Fortran and C implementations for some 35 computer platforms, as well as specialized libraries for graphics and parallel and distributed computing.

In the early 1970s, the NATS (National Activity to Test Software) group was established at Argonne Laboratory under government and university sponsorship to produce quality software for specific areas of numerical computation. Two packages were produced by this project, EISPACK for eigenvalue-eigenvector computation (1972) and FUNPACK for special function evaluation (1975). The software produced by the NATS effort was very well received, its high standards for performance, transportability, testing, certification, documentation, and dissemination establishing a paradigm for subsequent numerical software development efforts; see Table 1 for a partial list.

Table 1. Some PACKs for Numerical Computation¹

Name	Year ²	Size ³	Purpose
EISPACK	1972	70	Matrix eigenvalue problems
FISHPAK	1975	19	Separable elliptic partial differential equations
ELLPACK	1977	49	Elliptic partial differential equations
BLAS	1979	42	Elementary vector operations
LINPACK	1979	164	Matrix factorizations, linear systems, determinants, inverses
FNLIB	1979	204	Elementary and special functions
MINPACK	1980	10	Nonlinear systems and nonlinear least squares problems
FFTPACK	1982	18	Fast Fourier and related transforms
QUADPACK	1983	68	Numerical evaluation of one-dimensional integrals
ODEPACK	1983	6	Systems of ordinary differential equations
BLAS 2	1988	27	Elementary matrix-vector operations
BLAS 3	1990	48	Elementary matrix-matrix operations
LAPACK	1992	598	Numerical linear algebra
SCALAPACK	1995	26	Distributed numerical linear algebra

¹ Most of these packages are available from *netlib* at <http://www.netlib.org>.

² Date of first release.

³ Number of user-callable subprograms in 1999.

Excellent public domain packages such as these have greatly influenced program library development, and many of them have been incorporated into larger, more widely distributed libraries, both public domain and commercial. Their availability, in fact, was one of the prime motivations for the development of the SLATEC (Sandia National Laboratory Albuquerque-Los Alamos National Laboratory-Air Force Weapons Laboratory Technical Exchange Committee) library. (Sandia National Laboratory Livermore, Lawrence Livermore National Laboratory, the National Institute of Standards and Technology, and Oak Ridge National Laboratory were also partners in the project.) These groups wished to integrate this software into a new highly portable common mathematical library which would be (a) free of licensing restrictions for use within the laboratories, (b) supported jointly, thus reducing local library maintenance costs, (c) immediately available on newly acquired supercomputer systems, and (d) an aid to the interchange of application software among the labs. The committee, which was active from 1977 to 1993, solicited software for inclusion, and developed and enforced standards for portability, documentation, and testing. The library was highly successful, and is still widely used today.

Traditionally, the interface to mathematical and statistical libraries has been the procedure call, typically in Fortran. This requires that users write a calling program (usually) in the same language in which the library is coded. This is ideal when libraries are to be used within large complex applications. However, this method of interfacing with library routines is inconvenient for more casual users who are not necessarily expert in computer programming. This problem was recognized quite early. In the early 1960s the National Bureau of Standards developed a general purpose, interpretive, and portable program called OMNITAB which allowed high-level access to a varied collection of library procedures for statistical and numerical data analyses. A modern version of this program is still in use, and was the basis for the MINITAB system, now available commercially. The development of such high-level user interfaces continued to be important to the statistics community, and all of the original statistical libraries such as BMDP, SPSS, and SAS developed sophisticated user interfaces, including graphics and windows.

The move toward high-level user interfaces for *mathematical* libraries came somewhat later. The ELLPACK system for elliptic boundary value problems, developed in the late 1970s, provided a high-level problem-statement language to describe problems and invoke and compose procedures for solving them. During the same period, Cleve Moler of the University of New Mexico developed an interactive system called

MATLAB to ease use of the LINPACK and EISPACK libraries for students. A much enhanced commercial version is now marketed by The MathWorks. In 1988 Wolfram Research released Mathematica, a system which combined numerical, symbolic and graphical tools in a single product, appealing to sophisticated and neophyte users alike. These events set the stage for the large number of general-purpose mathematical computing systems providing high-level interfaces to rich underlying mathematical and statistical software libraries which are now available.

Issues

We next describe some of the issues which must be addressed in the development and maintenance of large numerical program libraries.

PORTABILITY

The costliness of the effort to adapt libraries to ever-changing computer systems has led to a concern for portability. Although most of the libraries described here are written in common programming languages like Fortran and C, not all can be moved easily from one computer to another; instead, libraries are often provided in a different implementation for each type of computer system. Several things stand in the way of portability: first, the dialect of the programming language in use and, second, the arithmetic differences among computers, both in static hardware and dynamic behavior resulting from differences in compiler-generated code.

Considerable progress has been made in overcoming these problems. Libraries are now usually programmed in standard Fortran or C, and their adherence to the standard can be mechanically verified. To cope with arithmetic differences among computers, machine-dependent code fragments are often flagged to be set by a preprocessor (*q.v.*) before compilation. An alternative technique is to have only one source code, with machine-dependent quantities obtained at run time using standardized function calls. The PORT library, developed by AT&T Bell Laboratories in 1974, pioneered this technique. In PORT, machine-dependent constants are obtained from three Fortran functions, R1MACH, D1MACH, and I1MACH. These routines are freely available (from *netlib*, for example; see below) and have become the basis of portability for a number of other libraries such as SLATEC.

For systems that provide high-level user interfaces to library routines there is also the added complication of interaction with services provided by the operating system, as well as windowing and graphics systems. These are not addressed here.

PERFORMANCE PORTABILITY

Modern computer architectures are complex and varied. Architectural details such as number of processors, their type (e.g. vector), type and size of cache (*q.v.*), page sizes (see VIRTUAL MEMORY), etc. can have profound effects on the efficiency of numerical algorithms. As a result, library developers have also become concerned not only with a library routine's ability to perform correctly, but with its *efficiency*, measured in terms of elapsed time and memory usage, as one moves the routine from one platform to another.

One promising approach to achieving what has come to be known as *performance portability* has been to encapsulate low-level, but compute-intensive, operations into standardized utilities which have good generic implementations, but which can be optimized for each platform. This was pioneered by the Basic Linear Algebra Subprograms (BLAS), released in 1979, which formed the basis for LINPACK. These low-level vector functions provided a degree of performance portability between scalar systems and early vector processors like the CRAY I and the Cyber 205. Modern symmetric multiprocessor systems and cache-based processors require the encapsulation of higher-level operations to achieve effective performance portability. This has led to the development of the Level 2 and Level 3 BLAS, which provide matrix-vector and matrix-matrix operations; these form the basis for LAPACK. Many computer manufacturers have provided optimized BLAS for their systems, and these are used in many libraries.

ERROR HANDLING

In order to protect users from program failure and from their own programming errors, the best quality program libraries do careful error checking. Both the legality of the input parameters to a subprogram and the validity of the computation process must be scrutinized. Some errors must be signaled as *fatal* whereas others can be designated as less serious. Unfortunately, no standard has been adopted for error handling, and procedures vary from one library to another. Within a given library, however, errors are most often reported through a fixed error handler. Users can often control the behavior of the handler as a function of the error severity. In some cases it is reasonable to print a notification of a fatal error and abort the program, while in other cases users need the flexibility to regain control of execution with an error flag set so that they can take appropriate action. Good error handlers provide both of these mechanisms.

DOCUMENTATION

A program library is of no use unless it is supported by documentation explaining how to use each program.

The purpose of the program, its input and output parameters, and possible error situations must be clearly described. Most libraries have detailed manuals which provide this information, as well as extensive background on the problems addressed.

Increasingly, library documentation is being kept online, permitting users to access the information interactively. In some cases, as with the SLATEC library, documentation is provided only in machine-readable form (*q.v.*). Because of this, SLATEC established rigid documentation standards for subprograms accepted into their library. The *SLATEC Prologue*, which is included in each subprogram, includes sections on purpose, problem classification, precision, keywords, authors, description, related routines, references, routines called, and revision history. Such standards greatly ease the integration of online documentation into local systems.

Many libraries now provide separate interactive online documentation systems employing keyword search, problem classifications, and decision trees to help users locate appropriate software. The National Institute of Standards and Technology Guide to Available Mathematical Software (GAMS) problem classification system is in widespread use for this purpose. This system is the basis for an online software advisory system which integrates information about software in more than 100 libraries and packages; it is available at <http://math.nist.gov/gams/>.

THE FUTURE

Many challenges remain in the development of numerical and statistical program libraries. Among these are massively parallel computers, new computer languages, and network-based computing.

Library developers have begun to address the problem of revising algorithms and interfaces to achieve improved performance on distributed memory multiprocessors. Providing high levels of performance as well as high levels of portability for such environments is especially difficult. The SCALAPACK library represents a first effort to do this for the solution of linear systems; it has formed the basis for several recent commercially supported libraries. The need to integrate mathematical problem solving into applications with very high computational demands is also straining the traditional black-box approach to the delivery of libraries. Such applications cannot afford to map their data into forms best suited to the numerical algorithm, and may need to exercise more control over the process. This is leading to the exploration of new techniques for the delivery of the expertise of numerical analysts to customers. Some facets of object-oriented computing (*q.v.*) may play an important role here.

Library users are increasingly moving to new languages like Fortran 90, C++, and Java to do their work. This is providing new demand for redesigning numerical and statistical libraries to adapt to the new computing paradigms underlying these languages. The pervasiveness of computer networks also opens the possibility of network-based libraries which provide remote access to problem solution services rather than source code for a local machine.

Finally, expansion of libraries into new mathematical and statistical problem domains as well as the incorporation of improved algorithms will continue indefinitely.

Program Library Availability

Table 2 provides current information on the accessibility of the large program libraries mentioned in this article.

Libraries classified as mathematical generally include evaluation of special functions, linear algebra, interpolation and approximation, solution of nonlinear algebraic equations, optimization, quadrature, solution of differential equations, integral transforms, and sorting (*q.v.*). Libraries classified as primarily statistical generally include data summarization, data manipulation, elementary data analysis, statistical function evaluation, random number generation (*q.v.*), analysis of variance, regression, categorical data analysis, time series analysis, and cluster analysis.

Of course, there are many libraries and sources of programs not represented here. This has become especially true in recent years as commercial ventures seek to exploit the booming market in personal computers and workstations (*q.v.*).

Journals in several scientific fields regularly publish algorithms and programs which form the basis for new library routines. For example, the Association for Computing Machinery began publishing refereed algorithms in their *Communications* in 1960, and the *Collected Algorithms of the ACM* contains algorithms published since that time. In 1975, publication of algorithms was transferred to the *ACM Transactions on Mathematical Software (TOMS)*. Similar libraries have appeared in related fields. In 1969 a library of codes whose descriptions were published in the journal *Computer Physics Communications* was established at the Queen's University of Belfast. Although primarily a library of software for physics, many refereed general-purpose mathematical and statistical codes have appeared there. Such numerical and statistical packages developed by the research community are readily available for downloading from the Internet (*q.v.*). Most of the packages listed in Table 1, for example, as well as much additional freely available software, can be

Table 2. Some libraries for numerical and statistical computation.

Name	Version	Size ¹	Area	Distributor
ACM CALGO	1997	775	Math, statistics	ACM, 1515 Broadway, 17th Floor, New York, NY 10036-5701. http://www.acm.org/calgo/
BMDP	Classic	40	Statistics	SPSS Inc., 444 N. Michigan Avenue, Chicago, IL 60611. http://www.spss.com
CPC	1997	1500	Math, statistics	CPC International Program Library, School of Maths and Physics, The Queen's University of Belfast, Belfast BT7 1NN, Northern Ireland, UK. http://www.cpc.cs.qub.ac.uk/cpc/
HARWELL	12	640	Math	AEA Technology plc, Culham Science Centre, Abingdon, Oxfordshire, OX14 3ED, UK. http://www.cse.clrc.ac.uk/Activity/HSL/
IMSL	3.0	960	Math, statistics	Visual Numerics, Inc., 9990 Richmond Ave., Houston, TX 77042. http://www.vni.com
Mathematica	3.0	N/A	Math, statistics	Wolfram Research, 100 Trade Center Drive, Champaign, IL 61820-7237. http://www.wri.com
MATLAB	5	N/A	Math, statistics	The MathWorks, Inc., 24 Prime Park Way, Natick, MA 01760-1500. http://www.mathworks.com
MINITAB	11	N/A	Statistics	Minitab Inc., State College, PA 16801. http://www.minitab.com
NAG	Mark 17	1295	Math, statistics	Numerical Algorithms Group Ltd., Wilkinson House, Jordan Road, Oxford OX2 8DR, UK. http://www.nag.co.uk
OMNITAB	7.0	N/A	Statistics	NTIS, US Dept. of Commerce, 5285 Port Royal Rd., Springfield, VA 22161. http://www.ntis.gov/fcpc/cpn5314.htm
PORT	3	373	Math	Lucent Technologies, Software Solutions Group, 150 Allen Road, Suite 2000, Liberty Corner, NJ 07938-1995. http://www.netlib.org/port/
SAS	5.18	N/A	Statistics	SAS Institute Inc., Box 8000, Cary, NC 27511-8000. http://www.sas.com
SLATEC	4.1	741	Math	netlib. http://www.netlib.org/slatec/
SPSS	2.1	N/A	Statistics	SPSS Inc., 444 N. Michigan Avenue, Chicago, IL 60611. http://www.spss.com

¹ Approximate number of user-callable program modules.

obtained from the *netlib* service of the University of Tennessee at Knoxville and Bell Laboratories (<http://www.netlib.org>). A similar collection of statistical software is maintained at Carnegie Mellon University (<http://lib.stat.cmu.edu>).

Acknowledgment

This article is a contribution of the National Institute of Standards and Technology (NIST) and is not subject to copyright. Certain commercial products are identified in the article in order to describe the history of program library development. Such identification does not imply recommendation or endorsement by NIST, nor does it imply that they are the best available for the purpose.

Bibliography

- 1951. Wilkes, M. V., Wheeler, D. J., and Gill, S. *The Preparation of Programs for an Electronic Digital Computer*. Reading, MA: Addison-Wesley.
- 1971. Rice, J. R. (ed.) *Mathematical Software*. New York: Academic Press.
- 1978. Fox, P. A., Hall, A. D., and Schryer, N. L. "Framework for a Portable Library," *ACM Transactions on Mathematical Software*, 4, 177-188.
- 1984. Cowell, W. R. (ed.) *Sources and Development of Mathematical Software*. Upper Saddle River, NJ: Prentice Hall.

- 1985. Boisvert, R. F., Howe, S. E., and Kahaner, D. K. "A Framework for the Management of Scientific Software," *ACM Transactions on Mathematical Software*, 11, 313-355.
- 1990. Dongarra, J. J., Du Croz, J., Hammarling, S., and Duff, I. "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Transactions on Mathematical Software*, 16, 1-17.

R. F. Boisvert

SOFTWARE MAINTENANCE

For articles on related subjects see COMPATIBILITY; DEBUGGING; ERRORS; OBJECT-ORIENTED PROGRAMMING; SOFTWARE CONFIGURATION MANAGEMENT; SOFTWARE ENGINEERING; and SOFTWARE PROJECT MANAGEMENT.

Introduction

The objective of *software maintenance* is to make required changes in software including changes to documentation (e.g. specification, design, listing, test plan) in such a way that its value to users is increased. Required changes can result from the need either to correct errors or to increase the capabilities of the software. Maintenance is not limited to making post-delivery changes. Rather, it starts with user requirements and continues for the life of the software.

TAB 3

Object-Oriented Information Engineering

Analysis, Design, and
Implementation

Stephen Montgomery



AP PROFESSIONAL
A Division of Harcourt Brace & Company

Boston San Diego New York
London Sydney Tokyo Toronto

This book is printed on acid-free paper. ∞

Copyright © 1994 by Academic Press, Inc.

All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission in writing from the publisher.

AP PROFESSIONAL

955 Massachusetts Avenue, Cambridge, MA 02139

An imprint of ACADEMIC PRESS, INC.

A Division of HAR COURT BRACE & COMPANY

United Kingdom Edition published by

ACADEMIC PRESS LIMITED

24-28 Oval Road, London NW1 7DX

Library of Congress Cataloging-in-Publication Data

Montgomery, Stephen.

Object-oriented information engineering: analysis, design, and implementation / Stephen Montgomery.

p. cm.

Includes bibliographical references and index.

ISBN 0-12-505040-2 (alk. paper)

1. Object-oriented databases. 2. Systems engineering. I. Title.

QA76.9.D3M649 1994

658.4'038'0285421—dc20

93-43044

CIP

Printed in the United States of America

94 95 96 97 98 ML 9 8 7 6 5 4 3 2 1

Advantages of LANs

An LAN allows microcomputers to share printers, plotters, network gateways and other peripheral devices. Services the network supplies are selected beyond simple device sharing to electronic mail, corporate database access and shared applications. Sharing applications and databases means that users of client machines do not need to swap diskettes in order to share information. Networked versions of microcomputer software often cost much less per user than stand-alone packages. Backup and restore operations on shared databases are much easier to manage with a shared server on a network. Also, the memory requirements of client workstations may be much less if a power server can handle intensive data management functions.

Network Management

Because of the trend to downsize large information systems to networks of interconnected smaller computers, management of network hardware and software will become much more important in the future. Network management needs to be planned in order to be most effective. Potential problems need to be detected and resolved, and recurring problems need to be addressed. Network modification and repair requests need to be tracked. Performance needs must be tracked and adjusted. Reports need to be generated and distributed. Setting up network devices and software utilities as objects can provide a network manager with a powerful means to manage "smart" network objects that send and receive messages to and from the operator.

17.4 CLIENT/SERVER ARCHITECTURE

Client/server computing involves servicing requests from one computer (the client) via a second computer (the server). This type of computing allocates certain computing functions to clients, usually on an LAN, and other functions to one or more servers, either on an LAN or on a larger network. A client may process application programs and the user interface and the server might be a terminal server, file server, database server, disk server, mail server, communications server or even a display server (for sophisticated graphics functions). Figure 17.3 depicts a client/server contract. Figure 17.4 lists client/server partnership characteristics.

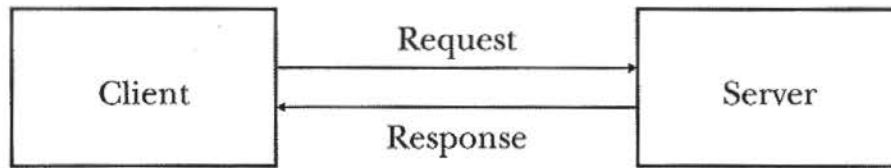


Figure 17.3 A client/server contract

The Client/Server Model

With a more traditional file server system, a client workstation has a database management system and user interface function resident in it. The file server contains file management software and certain files to be stored and managed. A client requests a file or files from the file server, which sends the contents of a file. The client database management software processes the file. The file server allows clients to share and manage files centrally and needs very fast disk drives and file access methods to be efficient. Figure 17.5 lists characteristics.

A client/server system, on the other hand, has the user interface in the client and only database requests are sent to a database server. All database processing is performed on the server machine, with only the necessary data records sent to the client (not entire files unless so requested).

Services

- A server is a provider of services
- A client is a consumer of services
- The client/server model provides a clean separation of function based on service

Shared resources

- A server can service many clients at the same time and regulate access to shared resources

Asymmetric protocols

- A many-to-one relationship exists between clients and a server
- Clients always initiate a dialog by requesting a service
- Servers passively wait for requests from clients

Figure 17.4 Client/server partnerships—characteristics of client/server architectures

Transparency of location

- A server is a process that can reside on the same machine as the client or on a different machine across a network
- Client/server software usually masks the location of the server from clients by redirecting service calls when needed

Hardware and operating system independence

- Ideally, client/server software is independent of hardware or operating systems
- There should exist the capability to mix and match client and server platforms

Message-based coupling

- Clients and servers are loosely coupled systems that interact through a message-passing mechanism
- A message is the way that service requests and replies are delivered

Encapsulation of services

- A server is a specialist
- A message tells a server what service is being requested; the server decides how to get the job done
- Services may be upgraded without affecting clients as long as the published message interface remains unchanged

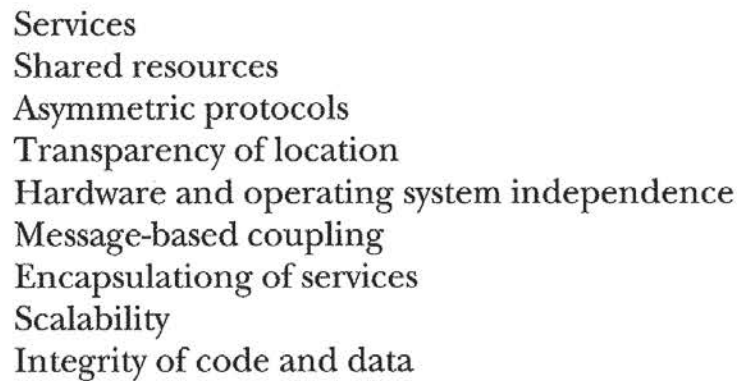
Scalability

- Client/server systems may be scaled horizontally or vertically
- Horizontal scaling refers to the addition or removal of client workstations with only a slight performance impact
- Vertical scaling refers to the migration to a larger and faster server machine or multiple servers

Integrity of code and data

- Server code and data are centrally maintained
- Maintenance is less costly
- Shared data integrity is maintained more easily

Figure 17.4 (continued) Client/server partnerships—characteristics of client/server architectures



- Services
- Shared resources
- Asymmetric protocols
- Transparency of location
- Hardware and operating system independence
- Message-based coupling
- Encapsulation of services
- Scalability
- Integrity of code and data

Figure 17.5 Client/server architectures—characteristics

Today, an implementation of the client/server model needs to have a user-friendly GUI as well as most or all of the application logic on the client. The client usually initiates actions. The client and a server must be distinct from each other yet they need to interact seamlessly. They need to be able to operate on different computing platforms but can reside on the same computer. A server needs to be capable of serving multiple clients simultaneously. A database server should provide functions such as security, backup, recovery, database query and data integrity protection. Finally, the system needs to be able to communicate over an LAN, WAN and other networks.

In order to write transparent, portable client/server applications, developers must be able to isolate application development from any one computing platform. Object-oriented development provides this platform independence via encapsulation and information hiding, and client/server development requires it. Three components are essential to any client/server application: data access, processing and interfaces. Data access includes the user interface and stored data access. Processing includes business processing logic. Interfaces link services with other applications. The separation of these components leads to a development of one component that is isolated from the other technology layers. Each layer isolates technology characteristics of that layer from the other layers.

Types of Client/Server Processes

Client/server architecture defines that each application is implemented in two parts: a client task (any task communicating with the user) and a server task (any other task that communicates with client tasks and/or other server tasks). Such an architecture has several components. Major services include

Typical clients:

- User window process
- Workstation application
- Host application

Types of servers:

- Application server
- Database server
- Transaction server
- File server

Typical servers:

- Print server
- Graphics server
- Mail server
- Communications server
- Network server

Figure 17.6 Typical client/servers

presentation, client, distribution, server and database functions. Figure 17.6 lists typical clients and servers. Figures 17.7 through 17.10 discuss the characteristics of some servers.

Presentation services include user interaction, memory management and user dialog control.

- A server may itself be a client for some system functions.
- An application may consist of numerous “contracts” involving many clients and many servers.
- A “contract” between objects consists of one or more “collaborations.” The contract between two objects is defined by the set of requests that a client can make of a server.
- A “collaboration” between objects consists of a stimulus (request) and a response (action).
- Collaborations and contracts can be modeled using state-transition models.

Figure 17.7 Client/server architectures—characteristics

- A client passes a request for file records over a network to a file server.
- This is a very primitive form of data service requiring many message exchanges over the network.

Figure 17.8 File servers—characteristics

- A client passes database requests as messages to a database server.
- Only the final results are returned to the client over the network.
- Database request code and data reside on the same machine.
- This allows the server to use its own processing power to find the requested data (no need to pass all records back to client, as with a file server).
- Server code is packaged by the DBMS vendor but code must often be written for client applications.

Figure 17.9 Database servers—characteristics

- A client invokes remote procedures that reside on a server, which also contains a database engine.
- Remote procedures on the server execute database calls as units of work (transactions).
- Applications are created by writing code for both client and server components.
- This is also referred to as on-line transaction processing (OLTP).
- Used for mission-critical applications requiring 1–3 second response all of the time.
- This requires tight controls over database security and integrity.
- Communication overhead is kept to a minimum—an exchange is a single request/reply pair (not multiple database calls).
- This requires a peer-to-peer protocol to issue calls to remote procedures and obtain results.

Figure 17.10 Transaction servers—characteristics

- Developers supply code for client and server.
- An application server is not necessarily database centered, unlike transaction servers.
- This is a general-purpose, customizable network application using the client/server model.

Figure 17.11 Application servers

Client services include:

Data capture

Client front-end functions (user interface event handling and error handling)

Shared data management

Session management

Security

Distribution services include:

Program-to-program communication

Message management

Protocol enforcement

Location of network resources

Server functions include:

Multiuser processing

Server front-end functions (messaging, recovery, error handling)

Security

Database isolation

Database services include:

Data integrity and recovery

Security

Accounting and chargeback

Performance tuning

Application isolation
Multiuser locking

Client tasks work with GUIs and need to format screen presentations and user actions in response, manage dialogs between systems and users and determine capabilities of servers. Clients also need to determine which server to send a task to. Different types of servers might be print servers, facsimile servers, database and file servers, computational servers, communications servers and application servers.

To gain the greatest benefit from client/server architectures, the environment should include such major technologies as:

- Distributed databases that support synchronized multiuser updates
- Application development tools for applications that work synchronously and dynamically as well as perform on heterogeneous process architectures
- Communications processes that operate independent of underlying network topologies

17.5 CLIENT CHARACTERISTICS

The Client's Role

As stated earlier, a client is primarily a user of services provided by one or more server tasks. The client/server model clearly separates functions based on client and server roles. A client (or server) may act as a client at one moment in time and then as a server at another moment in time. A client always will provide presentation services to users, probably in the form of a GUI. A GUI supports windows and allows a client user to conduct several simultaneous tasks in multiple sessions.

At present, Microsoft Windows and Macintosh system software do not support true multitasking—they only execute one task at a time in a communications session. OS/2 and the various versions of UNIX are preemptive multitasking operating systems and support any number of active communications sessions (limited only by physical memory constraints).

Client system software provides such facilities as Dynamic Data Exchange (DDE) and Object Level Embedding (OLE) that support cut-and-paste operations for spreadsheets, graphics and text between user windows. A client

TAB 4



US005546583A

United States Patent [19]

[11] Patent Number: 5,546,583

Shriver

[45] Date of Patent: Aug. 13, 1996

[54] METHOD AND SYSTEM FOR PROVIDING A CLIENT/SERVER INTERFACE IN A PROGRAMMING LANGUAGE

Shriver, David I., "Research on REXX in the CICS Environment", Share 77 Chicago, Illinois 1940, 1991, pp. 1-36.

[75] Inventor: David I. Shriver, Euless, Tex.

Primary Examiner—Thomas M. Heckler
Attorney, Agent, or Firm—David A. Mims, Jr.; L. Bruce Terry; Andrew J. Dillon

[73] Assignee: International Business Machines Corporation, Armonk, N.Y.

[21] Appl. No.: 223,276

[22] Filed: Apr. 5, 1994

[51] Int. Cl.⁶ G06F 13/00

[52] U.S. Cl. 395/650; 364/DIG. 1; 364/284.4

[58] Field of Search 395/650

[56] References Cited

U.S. PATENT DOCUMENTS

5,086,504	2/1992	Nemeth-Johannes et al.	395/700
5,255,386	10/1993	Prager	395/600
5,257,366	10/1993	Adair et al.	395/600
5,287,514	2/1994	Gram	395/700
5,291,585	3/1994	Sato et al.	395/500
5,317,722	5/1994	Evans	395/500
5,430,876	7/1995	Schreiber et al.	395/650

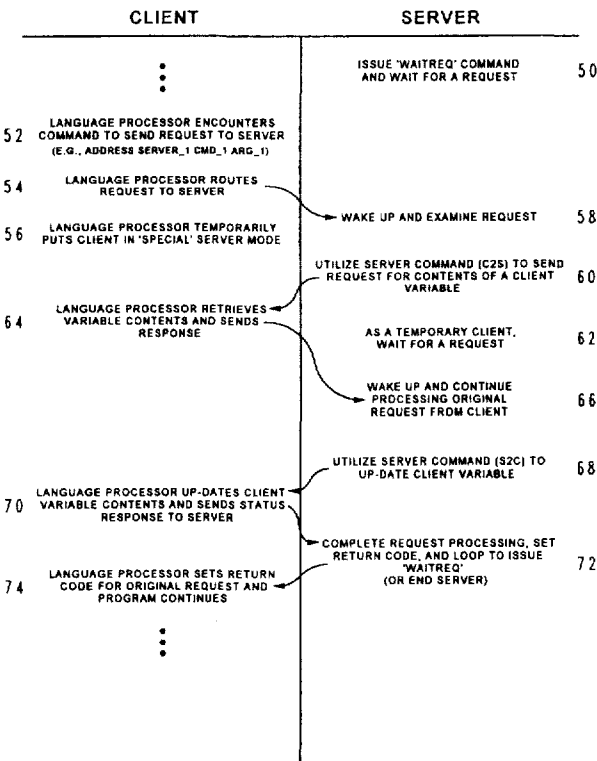
OTHER PUBLICATIONS

Shriver, David I., "REXX in the CICS Environment", Third REXX Symposium Annapolis, Maryland, 1992, pp. 1-41.
Shriver, David I., "Research on REXX in the CICS Environment", Share 80 San Francisco 1916, 1993, pp. 1-44.

[57] ABSTRACT

In a data processing system, a programming language processor capable of executing program code is provided. A client program and a server program are also provided within said data processing system. The client program and the server program are comprised of program code capable of execution within said data processing system. Once the client and server programs are invoked, the client program sends a request for a service to the server program. In response to program code within the server program, a request is sent to the client program for a service that requires access to a variable within the client program. The client program then processes the request from the server program and sends the server program a response. Thereafter, the server program continues processing the request from the client program in response to gaining access to the variable in the client program. If the server program has not been initialized when the client program requests a service, the client program automatically initializes the server program.

8 Claims, 6 Drawing Sheets



U.S. Patent

Aug. 13, 1996

Sheet 1 of 6

5,546,583

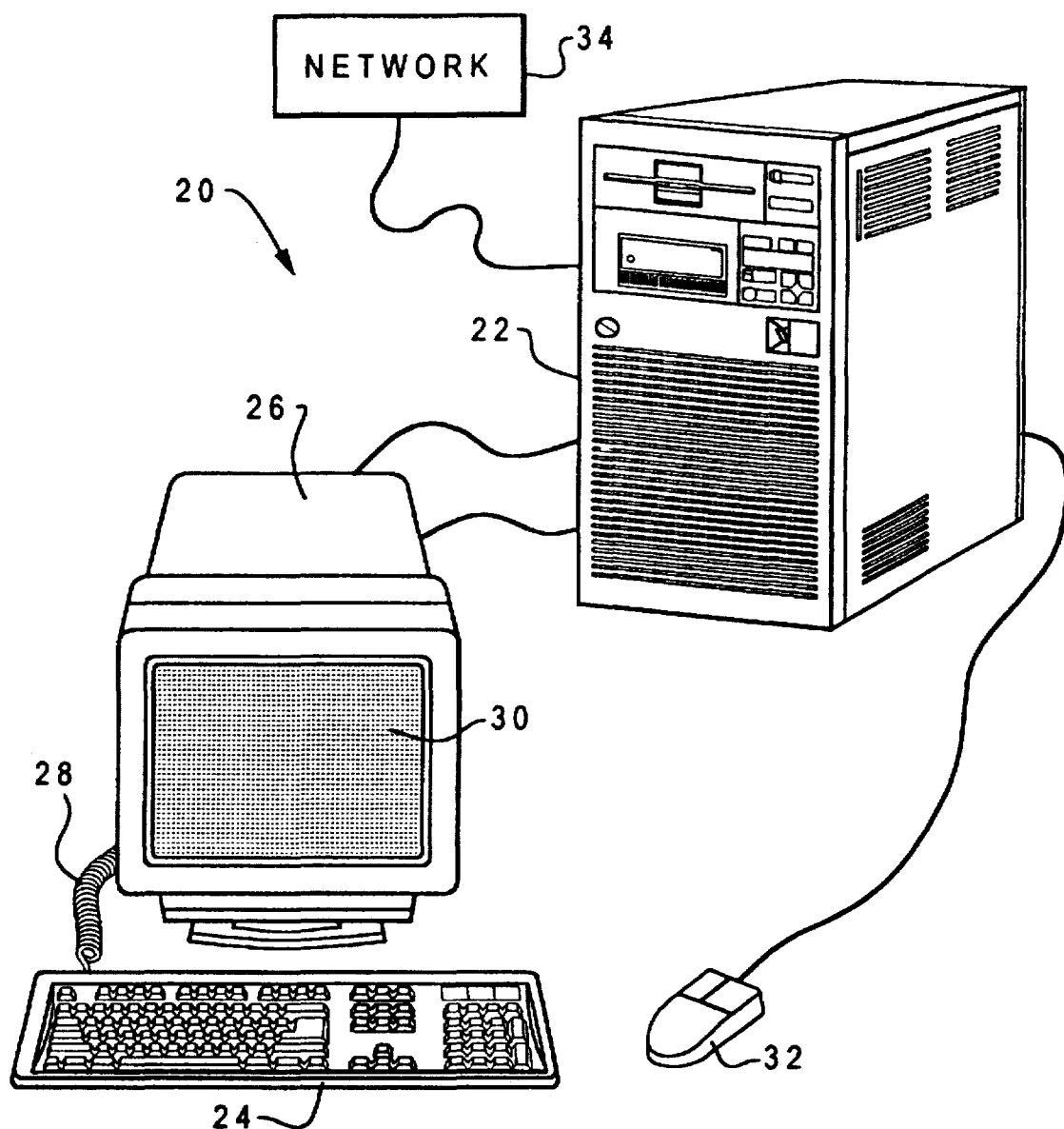


Fig. 1

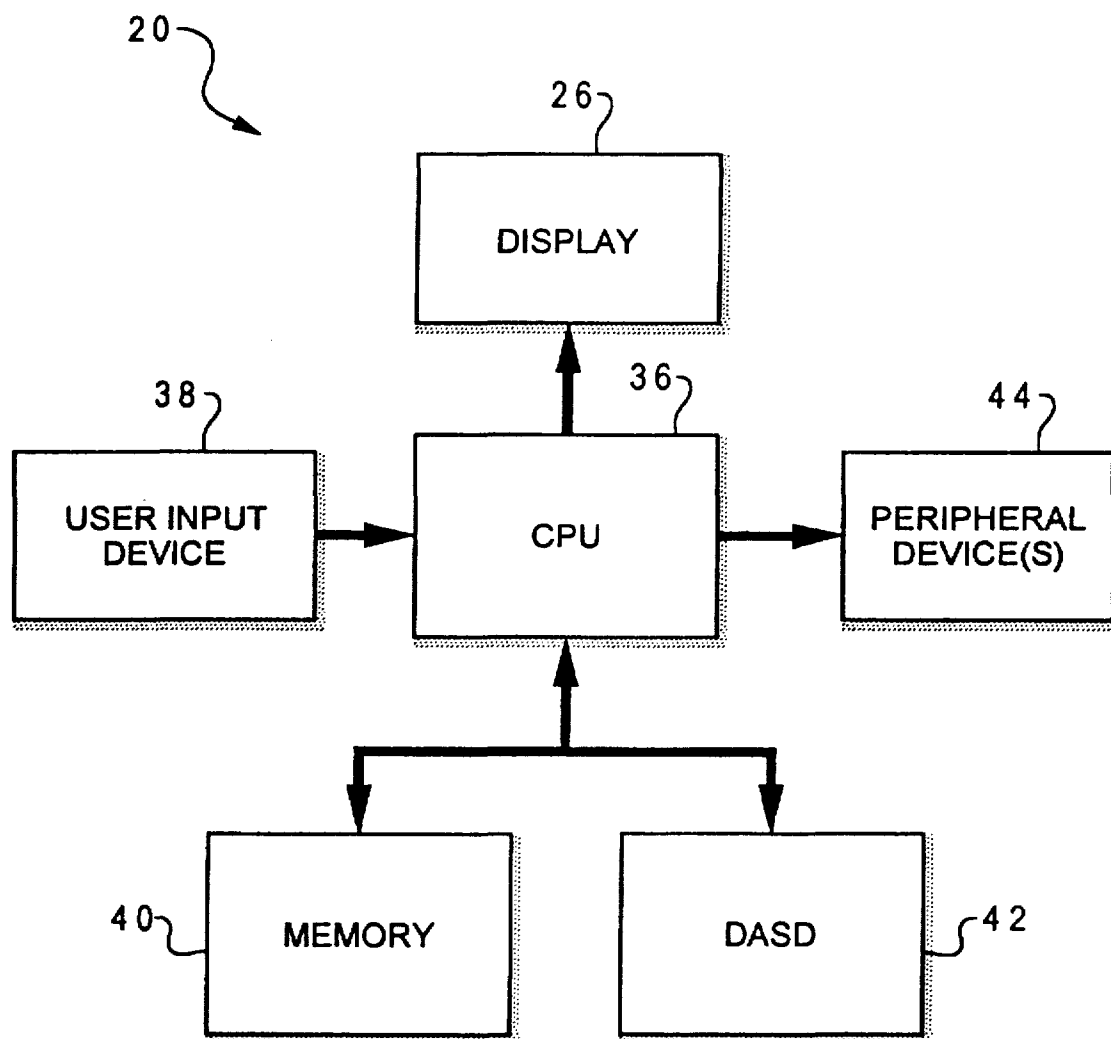


Fig. 2

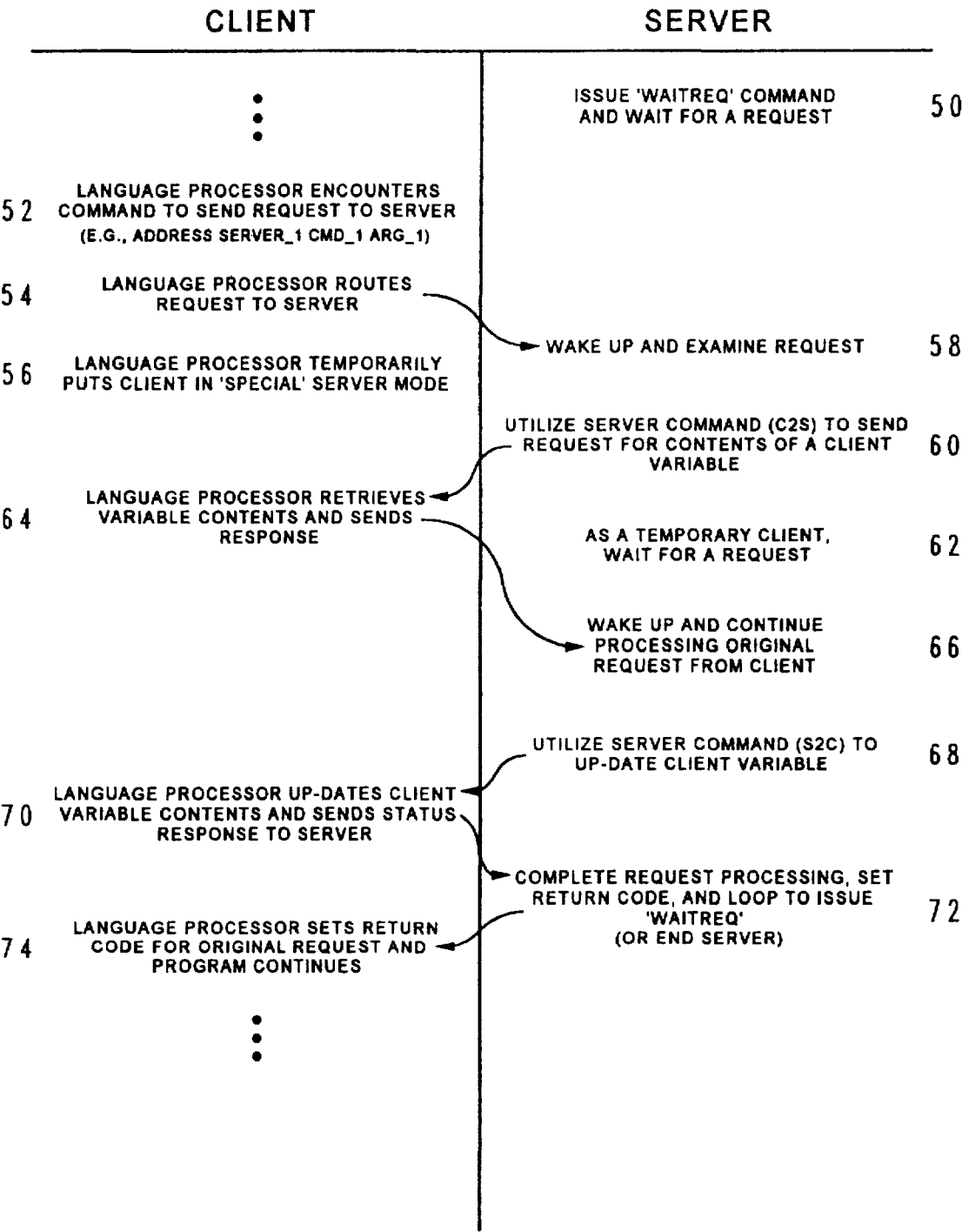


Fig. 3

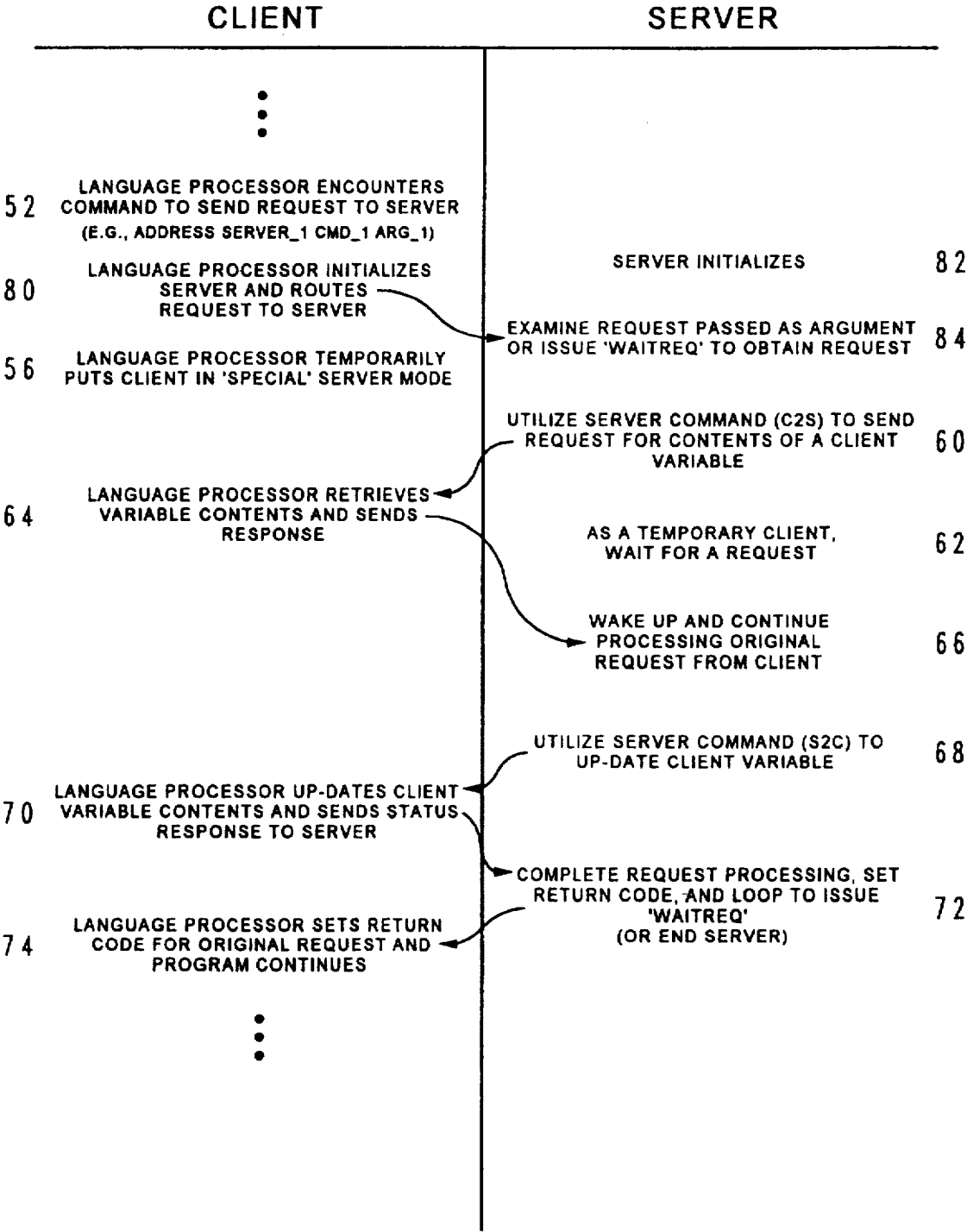


Fig. 4

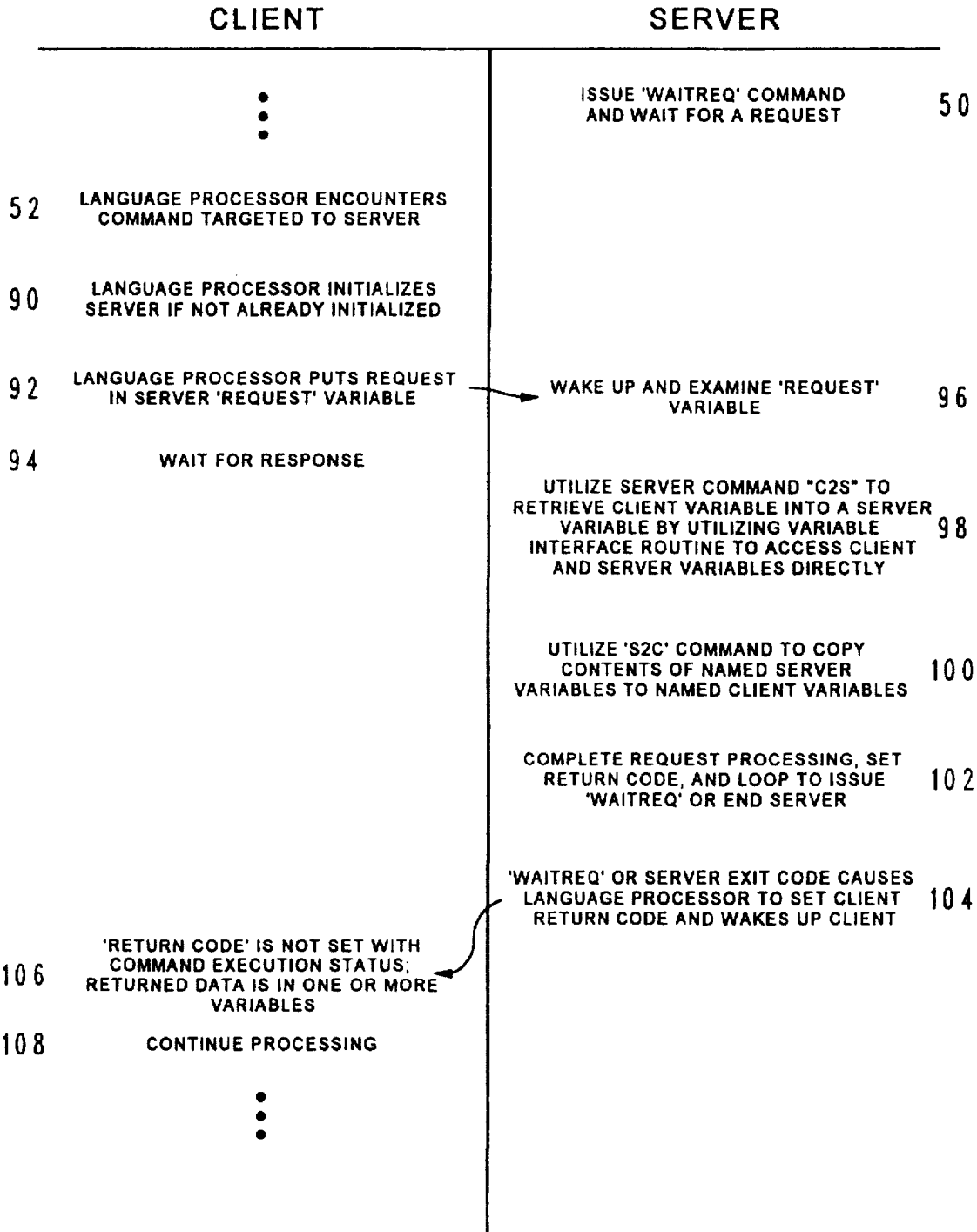


Fig. 5

U.S. Patent**Aug. 13, 1996****Sheet 6 of 6****5,546,583**

```

1      . . .
2
3      /* sample REXX client exec */
4      address SERVER2
5      'REQUEST1'
6
7      . . .
8
9
10
11
12
13     /* sample server exec */
14     DO FOREVER
15         'WAITREQ'      /* wait for request from client */
16         Select
17             When request = 'END' then LEAVE /* end the server */
18             When request = 'REQUEST1' then call Process_request1
19             Otherwise rc = -3 /* command not found */
20         end /* select */
21     END /* DO FOREVER */
22     exit
23
24
25     Process_request1:
26         vara = 100      /* set value to be passed to client */
27         'S2C VARA'      /* copy server variable to client variable */
28         rc = 0          /* send 'successful' return code to client */
29         return
30
31
32
33

```

Fig. 6

5,546,583

1

METHOD AND SYSTEM FOR PROVIDING A CLIENT/SERVER INTERFACE IN A PROGRAMMING LANGUAGE

BACKGROUND OF THE INVENTION

1. Technical Field

The present invention relates in general to an improved data processing system and in particular to a method and system for providing a client/server interface to a programming language. Still more particularly, the present invention relates to a method and system for providing client/server support which permits a server to wait for client requests and to read and set client variables.

2. Description of the Related Art

In the art of data processing system programming, "client/server" interaction is becoming an increasingly popular form of software architecture. As utilized herein, "client/server" interaction is a relationship between two software processes, which may be running on the same data processing system or on different data processing systems which are coupled via a network system. The "server" process is a provider of services. The "client" process is a consumer of services. Client/server interaction provides a clean separation of function between processes based on the idea of service. Some of the reasons for this increase in popularity are discussed below.

A server may service many clients at the same time and regulate their access to shared resources. Clients typically initiate the dialog between clients and servers by requesting a service. Servers typically wait passively for a request from a client.

The server may reside on the same machine as the client or on a different machine across a network. Within client/server software, the location of the server is usually masked or hidden from the client by redirecting the service calls when required. A program may be a client, a server, or both.

The ideal client/server software is independent of hardware or operating system software platforms. Users should be able to mix and match client and server platforms.

Clients and servers are loosely coupled systems which may interact through a message-passing mechanism. The message is the delivery mechanism for service requests and replies.

The server is a "specialist." A message tells a server what service is requested and it is then up to the server to determine how the service is accomplished. Servers may be upgraded without affecting the clients as long as the published message interface is not changed.

The server code and server data are typically centrally maintained, which results in cheaper maintenance and the guarding of shared data integrity. At the same time, clients remain personal and independent.

The client/server characteristics described above allow intelligence to be easily distributed across a network, and provide a framework for the design of loosely coupled network based applications.

Many data processing system manufacturers provide a programming language called a "procedures language" for use with their hardware platforms. For example, International Business Machines Corporation (IBM) of Armonk, N.Y. provides the REXX language as a system procedures language to be used with data processing systems that follow IBM's System Application Architecture (SAA). In addition to performing the procedures language role, REXX is often

2

used as a macro language, a simple application development language, as a prototyping language, and in personal computing. A procedures language may also be known as a glue language, a system extension language, an EXEC language, and a shell language.

Code written in a procedures language is often utilized to control the running environment, by providing an outer structure for what needs to be done or controlled. The role of a procedures language may be characterized by the fact that residual text is sent to a specified environment, such as the operating system's command line interface, for further interpretation. This includes system-specific commands, as well as commands which may be utilized to invoke whole applications. The procedures language allows programmers to write code that may be utilized to personalize their data processing system environments. Code written in procedures language may also allow access to other programs or data.

In the role of a macro language, the procedures language interfaces with the command line interface presented by an application to the application user. For example, the procedures language may interface with a text editor, such as XEDIT on VM or KEDIT on the PC. This allows the application user to utilize the procedures language to personalize the application by grouping together application commands in conjunction with procedures language logic and, if needed, system commands. Users may utilize a sequence of commands presented by the procedures language to the application to perform repetitive tasks, and extend the application user interface.

In the REXX procedures language, programs are constructed from the following basic components: (1) clauses or statements, (2) tokens, (3) expressions, (4) instructions, (5) assignments, and (6) separators. In a REXX program, each program line usually contains at least one clause or statement. REXX interprets one clause at a time. Each clause can be divided into tokens, which are typically separated by blanks. Every item delimited by blanks inside a clause is defined as a token. The special characters comma (","), semicolon (";") and colon (":") are also used as token delimiters.

An expression is composed of terms and operators. There are three varieties of terms: (1) strings, (2) symbols, and (3) function calls. Each operator acts on two terms (except for prefix operators, which act on the term following). There are four varieties of operators/operations: (1) string concatenation, (2) arithmetic operators, (3) comparative operators, and (4) logical (boolean) operators.

Instructions are identified by a REXX keyword or a group of REXX keywords specifying a particular task.

Assignments assign a variable a value. A variable is a symbol that may change value during the execution of a REXX program.

Separators are defined as special characters which indicate the ending or continuation of a clause. Normally each clause occupies one physical program line. However, multiple clauses on one line may be separated by a semicolon (";"). A clause spanning more than one line is continued with a comma (",") at the end of each line to be continued.

When a REXX program is interpreted, each clause is subjected to two processes: (1) translation and (2) execution. During translation, all comments are ignored and substitution occurs. If, during substitution, a token is identified as a variable, it is replaced by its value; variables not previously referenced are dynamically defined.

During execution, three types of clauses require action: (1) instructions which are recognized as REXX keywords

5,546,583

3

are executed; (2) assignments are made; and (3) commands are executed. In such execution of system commands, strings that are not recognized as null clauses, labels, assignments, or instructions, are passed to the calling environment for execution.

REXX is usable as a command or macro language for applications which have internal commands. It is also an extremely effective programming language, because it can utilize separately compiled program packages as REXX subprograms. These capabilities are facilitated by the "REXX environment model."

In the REXX environment model, there are essentially no illegal commands or statement forms. There may, however, be expressions which are illegal syntactically. Such command and statement forms can be categorized into those which are meaningful to the REXX interpreter and those which are not. Commands or statement forms which REXX does not understand are passed to the underlying environment in which REXX is executing, using the command interface defined for the environment.

When a REXX program is invoked, the calling environment, such as CMS or XEDIT, is the default environment in which commands will be executed. For example, an EXEC file invoked from XEDIT is passed on to CMS by XEDIT, thus making CMS the calling environment. If an expression is evaluated, resulting in a character string which does not represent a legitimate REXX operation, such a character string is submitted to the "addressed" environment for execution.

After execution of the command represented by such a character string, control is returned to the REXX interpreter after setting a return code appropriate for the addressed environment. This return code is assigned to a special REXX variable, "Rc." Upon return of control to the interpreter, the value of should be evaluated to determine whether or not alternative action is required. For example, if a command is intended to invoke a program stored in another file, and that file is not found, a return code will be returned indicating that the file was not found. Typically, successful completion of a command results in RC set to zero.

At present, if a user wishes to implement client/server support in the REXX procedures language, the user may add new interface routines as function calls or as callable sub-routines, both of which may require code written in a language other than REXX. On the client side, the calls should be able to identify the target server and to pass a request. On the server side, routines should be able to wait for requests from a client, and retrieve and set client variables. At present, some procedural languages, such as REXX, do not have instructions for providing a "wait-for-request" function. Nor are instructions for reading and setting variables in a client program from a server program available in some procedural languages.

When adding new instructions to a programming language, the naturalness, readability, and maintainability (for which REXX is known) should be maintained.

Therefore, the problem remaining in the prior art is to provide the user of a programming language a method and system for providing client/server support which permits a server to wait for client requests and to read and set client variables, in a manner consistent with the programming language's friendly "look and feel."

SUMMARY OF THE INVENTION

It is therefore one object of the present invention to provide an improved data processing system.

4

It is another object of the present invention to provide a method and system for providing a client/server interface to a programming language.

It is yet another object of the present invention to provide a method and system for providing client/server support which permits a server to wait for client requests and to read and set client variables.

It is yet another object of the present invention to provide the user of a programming language a method and system for providing client/server support that allows users to benefit from the programming language's friendly "look and feel."

The foregoing objects are achieved as is now described. In a data processing system, a programming language processor capable of executing program code is provided. A client program and a server program are also provided within said data processing system. The client program and the server program are comprised of program code capable of execution within said data processing system. Once the client and server programs are invoked, the client program sends a request for a service to the server program. In response to program code within the server program, a request is sent to the client program for a service that requires access to a variable within the client program. The client program then processes the request from the server program and sends the server program a response. Thereafter, the server program continues processing the request from the client program in response to gaining access to the variable in the client program. If the server program has not been initialized when the client program requests a service, the client program automatically initializes the server program.

The above as well as additional objects, features, and advantages of the present invention will become apparent in the following detailed written description.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself however, as well as a preferred mode of use, further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

FIG. 1 depicts a data processing system which may be utilized to implement a preferred embodiment of the present invention;

FIG. 2 is a more detailed high-level block diagram further illustrating the major components of the data processing system of FIG. 1;

FIG. 3 depicts the interaction during program execution between a client program and an initialized server program, in accordance with the method and system of the present invention;

FIG. 4 illustrates the interaction during program execution between a client program and an uninitialized server program, in accordance with the method and system of the present invention;

FIG. 5 depicts the interaction during program execution between a client program and a server program, which are both located within a single data processing system, in accordance with the method and system of the present invention; and

FIG. 6 is a program code sample which illustrates the process of sending a request from a client to a server, and

5,546,583

5

passing variables between a client and server in accordance with the method and system of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENT

With reference now to the figures and in particular with reference to FIG. 1, there is depicted a data processing system 20 which includes processor 22, keyboard 24, and display 26. Keyboard 24 is coupled to processor 22 via cable 28. Display 26 includes display screen 30, which may be implemented utilizing a cathode ray tube (CRT), a liquid crystal display (LCD), an electroluminescent panel, or the like. Data processing system 20 also includes pointing device 32 which may be implemented utilizing a trackball, joystick, touch sensitive tablet or screen, or as illustrated in FIG. 1, a mouse. Pointing device 32 may be utilized to move a pointer or cursor on display screen 30. Those persons skilled in the art of data processing system design will recognize that display 26, keyboard 24, and pointing device 32 may each be implemented utilizing any one of several known off-the-shelf components. Data processing system 20 may be implemented utilizing any so-called "personal computer," such as the personal computer sold under the trademark "PS/2" which is manufactured and distributed by International Business Machines Corporation (IBM), of Armonk, N.Y.

With reference now to FIG. 2, there is depicted a more detailed high-level block diagram further illustrating the major components of data processing system 20 of FIG. 1. Data processing system 20 is controlled primarily by instructions, in the form of software, executed within central processing unit (CPU) 36. CPU 36 is coupled to display 26, which is utilized to display text and graphics, and possibly animated graphics or video. CPU 36 is also coupled to user input device 38, which is utilized to receive input from a data processing system user. User input device 38 may include keyboard 24 and pointing device 32, as illustrated in FIG. 1. Memory 40 and direct access storage device (DASD) 42 may be utilized for storing application programs (i.e., software) and data sets.

Peripheral devices 44 may also be included in data processing system 20. Such peripheral devices may include communications devices (i.e., modems or network adapters), or an audio output device for use during a multimedia presentation.

Referring now to FIG. 3, there is depicted the interaction during program execution between a client program and an initialized server program, in accordance with the method and system of the present invention. As illustrated by the ellipsis in the client column, the client program has been running before the steps illustrated in this figure occur, and the client program continues to run after the steps depicted have occurred.

The server program, which may be written in REXX program code, has been initialized and is running before a request is received from the client, as illustrated at reference numeral 50. To initiate a request by the client program, the programmer specifies in program code an environment, a command, and the appropriate command arguments. Such a program code instruction may take the form:

ADDRESS SERVER_1 CMD_1 ARG_1; where the environment is "SERVER_1," the command is "CMD_1," and the argument passed is "ARG_1."

During execution of the client program, the language processor encounters the above instruction and, after parsing

6

the instruction tokens, sends a request string (i.e., CMD_1 ARE_1) to the server program, as depicted at reference numerals 52 and 54. The language processor may utilize tables to look up the location (i.e., the server-ID) of the server as specified by the environment selected by the "ADDRESS" instruction, the intended, and the specific command. Once the command processor determines that the target of a command is a server, standard client/server support routines are invoked. The server may be located within the same data processing system, or another data processing system coupled via any one of several known networking systems.

Before receiving the request, the server is "waiting." By utilizing a command, such as the "WAITREQ" command, the server may be instructed to wait for a request from a client, as illustrated at reference numeral 50. This "WAITREQ" command causes the server exec to be suspended until a request from a client arrives.

After the client sends the request, the client program temporarily enters a "server" mode, as illustrated at reference numeral 56. While in such a "server" mode, the client is suspended, except that the client may receive and process a request from the server.

After receiving the request, the server wakes up (i.e., discontinues the "WAITREQ" instruction) and examines the request, as depicted at reference numeral 58. To process the client request, program code in the server may need information contained in a variable in the client program. If this is the case, the server may request the contents of a client variable by utilizing the "C2S" command to make such a request to the client program, as illustrated at reference numeral 60. If the client variable requested is not an existing server variable, a new variable by the same name or, optionally, a new name is created in the server program.

After making a request for client variable contents, the server temporarily enters a "client" mode and waits for the response to the request, as depicted at reference numeral 62.

Upon receiving a request from the server for the contents of a client variable, the language processor that executes the client program retrieves the client variable contents and sends the contents, in the form of a response, to the server, as illustrated at reference numeral 64.

This response from the client to the server causes the server to wake up and continue processing the original request from the client with the benefit of having the contents of a selected client variable, as depicted at reference numeral 66.

If the server calculates a new value for the contents of the requested client variable, the server may up-date the client variable by utilizing the "S2C" command, as illustrated at reference numeral 68. In response to executing the "S2C" command, the language processor sends a request to the client. When such a request is received by the client, the language processor executing the client program writes, or up-dates, the contents of the client variable, and then sends a status response to the server, as depicted at reference numeral 70.

Upon receiving the status response from the client, the server completes the processing of the original request, sets an appropriate return code, and loops or jumps to the "WAITREQ" command in the server code to wait for the next request for service, as illustrated at reference numeral 72. Such a return code is utilized to indicate the status of the execution of the service by the server.

As an alternative to looping to issue another "WAITREQ" command, the server may "end." If the server "ends," execution of server program code is discontinued. After a

5,546,583

7

server "ends," the server must be reinitiated before the server is able to provide a service in response to a new request.

When the server is finished processing the client request, the language processor executing the client program communicates the server's return code to the client program, and client program continues, as depicted at reference numeral 74.

With reference now to FIG. 4, there is depicted a chart which illustrates the interaction during program execution between a client program and an uninitialized server program, in accordance with the method and system of the present invention. Many steps in FIG. 4 are the same as corresponding steps in FIG. 3, and therefore, such steps have been identified with the same reference numerals.

FIG. 4 differs from FIG. 3 in that the server program is automatically initialized or initiated or invoked by the language processor running the client program when the client program sends a request to the server, as illustrated at reference numerals 80 and 82. After the server initializes, the server may examine the request and arguments passed from the client, or the server may begin waiting if such a request and arguments are not present, as depicted at reference numeral 84.

Thereafter the processing and interaction between client and server proceeds as described in reference to FIG. 3.

With reference now to FIG. 5, there is depicted a chart which illustrates the interaction during program execution between a client program and a server program which are both located within the same data processing system, in accordance with the method and system of the present invention. As illustrated, the server may be initiated before a request for service is received, as illustrated at reference numeral 50, or the language processor will initiate the server program when needed, as depicted at reference numeral 90.

Whether the server has been initiated or not, after the language processor has encountered a command targeted to a server (see reference numeral 52), the language processor puts the request for service in the server "request" variable, as illustrated at reference numeral 92. Thereafter, the client waits for a response, as depicted at reference numeral 94. In this case, the client is not temporarily put into a "special" server mode as described above with reference to FIGS. 3 and 4 above, because, in this implementation, the server may access the client's variables directly.

When the clients sets the server "request" variable, the server wakes up and examines the "request" variable, as illustrated at reference numeral 96. If the server needs the contents of a client variable, the server may utilize the "C2S" command to retrieve the contents of a client variable into a server variable, as depicted at reference numeral 98. Such a retrieval of the contents of a client variable is accomplished utilizing a variable interface routine which permits access to client and server variables directly. In the REXX language, for example, such a variable interface routine is called "EXECComm".

In a similar manner, the server program may utilize the "S2C" command to copy contents of named server variables to named client variables, as illustrated at reference numeral 100. When either the "S2C" or "C2S" command is utilized, the variables in the requesting client are automatically addressed.

After the server has retrieved and set the necessary client variables, the server completes processing the request, sets the return code, and loops to the "WAZTREQ" command near the beginning of the server program, as depicted at reference numeral 102. As an alternative to looping to the

8

"WAITREQ" command, the server may end, as discussed above.

As the server returns to the "WAITREQ" command, or as the server ends, the language processor sets the client return code and wakes up the client, as illustrated at reference numeral 104. The client server then receives returned data in one or more variables, as depicted at reference numeral 106. At this step, the return code is not set to indicate command execution status.

Thereafter, the client continues processing, as illustrated at reference numeral 108.

Finally, with reference to FIG. 6, there is depicted a program code sample which illustrates the process of sending a request from a client to a server, and the process of passing variables between a client and server, in accordance with the method and system of the present invention. Lines one through seven illustrates code that may be utilized in a client program to send a request to a server program. Lines 13 through 29 illustrates code that may be utilized in a server program.

As illustrated in line four, the program code sets the environment wherein the desired server is located. In line five, the command "request1" is sent to the environment "server2."

The server exec code is depicted at lines 13 through 29. In line 15, utilization of the "WAITREQ" instruction is illustrated. Server execution is suspended at line 15 until a "request" is received. When a request is received, the program is directed by instructions in lines 17 through 19.

For example, if the request variable equals "END," the process ends the server, at line 22. If the request variable equals "END," the process calls the subroutine "Process_request1" located at line 25. If the request variable is equal to some other string, the server returns a return code (e.g., rc=-3) which indicates the command was not found in the server.

In the subroutine "Process_request1" located at line 25, the program first sets the value of variable "VARA" equal to "100". In line 27, the program utilizes the "S2C" command to copy the server variable "VARA" to the client variable "VARA". If the client variable "VARA" is not an existing client variable, a new variable by the same name or, optionally, a new name is created in the client program. Thereafter, in line 28, the return code "rc" is set to "0" to indicate the command was successfully completed. In line 29, the program returns to the "do forever" loop at line 14 and waits for another service request at the "WAITREQ" command in line 15.

In the description of the present invention, examples of synchronous client requests have been illustrated. A request by a client is synchronous when the client suspends further processing until the server responds to the request. Although the embodiments of the present invention have been described with reference to synchronous client requests, programming languages may also include support for asynchronous client requests by providing queuing and the ability to test for a response or wait for a response. Such testing or waiting for a response may be implemented by utilizing a correlation ID passed along with the original request.

While the invention has been particularly shown and described with reference to a preferred embodiment, it will be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention.

5,546,583

9

What is claimed is:

1. A method in a data processing system for providing client/server interaction capability in a programming language processor for executing program code, said method comprising the steps of:

providing a programming language processor within said data processing system;

providing a client program within said data processing system wherein said client program is comprised of program code for executing within said data processing system;

providing a server program within said data processing system wherein said server program is comprised of program code for executing within said data processing system;

invoking said client program;

invoking said server program;

from said client program, sending a request for a service to said server program;

within said server program, sending a request to said client program for a service that requires access to a variable within said client program, wherein said request to said client program by said server program is sent in response to program code within said server program;

within said client program, processing said request from said server program; and

within said server program, processing said request from said client program in response to said access to said variable in said client program.

2. The method in a data processing system for providing client/server interaction capability in a programming language processor according to claim 1 wherein said step of sending a request to said client program for a service that requires access to a variable within said client program includes sending a request to said client program for a service that requires reading a variable within said client program.

3. The method in a data processing system for providing client/server interaction capability in a programming language processor according to claim 1 wherein said step of sending a request to said client program for a service that requires access to a variable within said client program includes sending a request to said client program for a service that requires modifying a variable within said client program.

4. The method in a data processing system for providing client/server interaction capability in a programming language processor according to claim 1 wherein said step of invoking said server program includes automatically invoking said server program utilizing said programming language processor.

10

5. A data processing system for providing client/server interaction capability in a programming language processor for executing program code comprising:

a programming language processor within said data processing system;

a client program within said data processing system wherein said client program is comprised of program code for executing within said data processing system;

a server program within said data processing system wherein said server program is comprised of program code for executing within said data processing system;

means for invoking said client program;

means for invoking said server program;

means for sending a request for a service from said client program to said server program;

means for sending a request in response to program code within said server program to said client program from said server program for a service that requires access to a variable within said client program;

means in said client program for processing said request from said server program; and

means in said server program for processing said request from said client program responsive to said access to said variable in said client program.

6. The data processing system for providing client/server interaction capability in a programming language processor according to claim 5 wherein said means for sending a request in response to program code within said server program to said client program from said server program for a service that requires access to a variable within said client program includes means for sending a request to said client program for a service that requires reading a variable within said client program.

7. The data processing system for providing client/server interaction capability in a programming language processor according to claim 5 wherein said means for sending a request in response to program code within said server program to said client program from said server program for a service that requires access to a variable within said client program includes means for sending a request to said client program for a service that requires modifying a variable within said client program.

8. The data processing system for providing client/server interaction capability in a programming language processor according to claim 5 wherein said means for invoking said server program includes means for automatically invoking said server program utilizing said programming language processor.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 5,546,583
DATED : August 13, 1996
INVENTOR(S) : Shriver

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 6, line 2: replace "ARe_1)" with --ARG_1--

Column 6, line 63: replace "7;" with --72--

Column 7, line 65: replace "WAZTREQ" with --WAITREQ--

Column 8, line 48: replace "lccp" with --loop--

Signed and Sealed this
Nineteenth Day of November, 1996

Attest:



BRUCE LEHMAN

Attesting Officer

Commissioner of Patents and Trademarks

TAB 5

A Dictionary of
Computing

FOURTH EDITION

Oxford New York

OXFORD UNIVERSITY PRESS

1997

Oxford University Press, Great Clarendon Street, Oxford OX2 6DP
Oxford New York
Athens Auckland Bangkok Bogota Bombay Buenos Aires
Calcutta Cape Town Dar es Salaam Delhi Florence Hong Kong
Istanbul Karachi Kuala Lumpur Madras Madrid Melbourne
Mexico City Nairobi Paris Singapore Taipei Tokyo Toronto Warsaw
and associated companies in
Berlin Ibadan

Oxford is a trade mark of Oxford University Press

© Market House Books Ltd. 1983, 1986, 1990, 1996

First published 1983

Second edition 1986

Third edition 1990

Fourth edition 1996

First issued (with corrections) as an Oxford University Press paperback 1997

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, without the prior permission in writing of Oxford University Press. Within the UK, exceptions are allowed in respect of any fair dealing for the purpose of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act, 1988, or in the case of reprographic reproduction in accordance with the terms of the licences issued by the Copyright Licensing Agency. Enquiries concerning reproduction outside these terms and in other countries should be sent to the Rights Department, Oxford University Press, at the address above

This book is sold subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out or otherwise circulated without the publisher's prior consent in any form of binding or cover other than that in which it is published and without a similar condition including this condition being imposed on the subsequent purchaser

British Library Cataloguing in Publication Data

Data available

Library of Congress Cataloging in Publication Data

Data available

ISBN 0-19-280046-9

1 3 5 7 9 10 8 6 4 2

Printed in Great Britain by

Biddles Ltd

Guildford and King's Lynn

program decomposition The breaking down of a complete program into a set of component parts, normally called modules. The decomposition is guided by a set of design principles or criteria that the identified modules should reflect. Since the decomposition determines the coarse structure of the program, the activity is also referred to as *high-level* or *architectural design*. See also modular programming, program design.

program design The activity of progressing from a specification of some required program to a description of the program itself. Most phase models of the *software life cycle recognize program design as one of the phases. The input to this phase is a specification of what the program is required to do. During the phase the design decisions are made as to how the program will meet these requirements, and the output of the phase is a description of the program in some form that provides a suitable basis for subsequent implementation.

Frequently the design phase is divided into two subphases, one of coarse *architectural design* and one of *detailed design*. The architectural design produces a description of the program at a gross level; it is normally given in terms of the major components of the program and their interrelationships, the main algorithms that these components employ, and the major data structures. The detailed design then refines the architectural design to the stage where actual implementation can begin. See also program design language.

program design language (PDL) A language, used for expressing *program designs, that is similar to a conventional high-level programming language but emphasizes structure and intention rather than the ability to execute programs expressed in the language. PDLs are often employed in conjunction with *structured programming. When not executable they are termed *pseudolanguages.

Typically the formal syntax of a PDL would cover data definition and overall program structure. Facilities in the latter area would include the basic control-flow constructs – sequential, conditional, and itera-

tive – plus those for the definition and invocation of subroutines. These facilities would be used to define the overall framework of the program, but individual actions within the framework would be expressed using pseudolanguage – natural English mixed with a more formal semantically rich language. Correspondingly, the PDL facilities for data definition may be expected to be richer than those of a typical programming language, encompassing a broader range of basic types and a more extensive set of data-structuring facilities. A wide variety of PDLs have been defined; normal practice is to select one that is well-matched to the target programming language.

program development system A software system that provides support to the program development phase of a software project. A typical program development system employs a simple database (or perhaps just a basic filing system) as a repository for information, and offers *software tools for editing of program source texts, compiling, link loading, and debugging. Usually some form of command-language interpreter is also available; this may have been produced specifically for the program development system, or may have been inherited from the underlying operating system. Compare software engineering environment.

program file A *file containing one or more programs, or program fragments, in *source code or *object code form.

program library (software library) A collection of programs and packages that are made available for common use within some environment; individual items need not be related. A typical library might contain compilers, utility programs, packages for mathematical operations, etc. Usually it is only necessary to reference the library program to cause it to be automatically incorporated in a user's program. See also DLL.

program listing (source listing; listing) An output produced by a *compiler or *assembler, consisting of the source program neatly laid out and accompanied by diagnostic information and error messages. In the

CERTIFICATE OF SERVICE

I hereby certify that, on this 28th day of March, 2016, I filed the foregoing Plaintiff-Cross Appellant Apple Inc.'s Combined Petition for Panel Rehearing and Rehearing En Banc with the Clerk of the United States Court of Appeals for the Federal Circuit via the CM/ECF system, which will send notice of such filing to all registered CM/ECF users.

/s/ William F. Lee

WILLIAM F. LEE
WILMER CUTLER PICKERING
HALE AND DORR LLP
60 State Street
Boston, MA 02109
(617) 526-6000

March 28, 2016

CERTIFICATE OF COMPLIANCE

Counsel for Plaintiff-Cross Appellant Apple Inc. hereby certifies that:

1. The brief complies with the type-volume limitations of Federal Rule of Appellate Procedure 40(b) because exclusive of the exempted portions it does not exceed 15 double-spaced pages.

2. The brief complies with the typeface requirements of Federal Rule of Appellate Procedure 32(a)(5) and the type-style requirements of Federal Rule of Appellate Procedure 32(a)(6) because it has been prepared using Microsoft Word 2010 in a proportionally spaced typeface: Times New Roman, font size 14 point.

/s/ William F. Lee

WILLIAM F. LEE

WILMER CUTLER PICKERING

HALE AND DORR LLP

60 State Street

Boston, MA 02109

(617) 526-6000

March 28, 2016